Escola Tècnica Superior d'Enginyeria Industrial de Barcelona

Universitat Politècnica de Catalunya

MASTER'S DEGREE IN AUTOMATIC CONTROLS AND ROBOTICS

Using Symmetries in Reinforcement Learning of Bimanual Robotic Tasks

Author:
Fabio AMADIO

Director:
Dr. Adrià COLOMÉ FIGUERAS

Speaker:
Prof. Maria SERRA PRAT

Barcelona, 18 June 2018

Contents

Introduction

Nowadays, the number and variety of robots used in everyday life and production are rapidly increasing. Most common applications are relegated in the industrial world, where robots stand in the assembly line, executing repetitive works, requiring a high degree of precision, with the aim of increasing the production rate and reducing human presence in dangerous environments. Outside of the factory, robotic applications in the market are currently able to handle only simple tasks with a low degree of interaction with human beings. One of the new challenge in robotics is to design autonomous robots able to share space with us without the need to modify the infrastructure of our home environment, in contrast to standard industrial robots, thought to work in predefined factory settings forbidden to humans. This vision creates a whole set of new collaborative and interaction opportunities, but these new environments are also exposed to various sources of perturbation and unpredictable situations not considered before, to which the robot needs to adapt. In *Robot Learning*, machine learning methods are used to automatically extract relevant information from data to solve robotic tasks. Modern machine learning techniques, with their power and flexibility, can help to further automate robotics and to narrow the gap towards fully autonomous robots, e.g., for general assistance in households, elderly care, and public services. It becomes crucial in this context to provide these robots with the capacity of generalizing movements and skills, since the range of possible tasks that they could carry out while cooperating with humans is in principle infinite. One of the key requirements is to make available user-friendly ways of programming robots to "teach" new skills and adapt existing ones to new situations. This possibility is described by *Learning from Demonstration* [1] paradigm, that provides users with a way to teach robots new knowledge by simply showing some demonstrations, generally by moving the robot himself as it should do to correctly execute the task (kinesthetic teaching). In future perspective, this kind of approach gives to users an instrument to adapt the robot to their needs, regardless of their computer programming skills or experience in robotics.

State of the art

The state-of-the-art method for learning robot movements is *Policy Search* [1], that uses parameterized policies π_θ operating directly in the parameter space Θ, $\theta \in \Theta$, while performing *Reinforcement Learning*. The set of parameters representing motion is updated iteratively from robot executions, evaluated with a reward function measuring the effectiveness in solving the desired task. The initial policy can be obtained from some demonstrated movements showed by the user before the start of the learning process (learning from demonstration). At each iteration, a different policy is tried, exploring the space of parameters according to the characteristics of the specific algorithm adopted. The reward values obtained are used to update parameters at each step, until convergence of reward function is reached. Policy search allows task-appropriate pre-structured policies, such as *Movement Primitives* (MPs) [3]. MPs are formulations

that gives a compact representation of the robot's policy allowing to deal with the inherently continuous robot movements. They generally depend linearly on a set of parameters (that are the weights of linear basis function models), whose modulation can adapt MPs to any trajectory, resulting in a highly flexible framework for robot learning. Policy search approach have demonstrated its effectiveness in numerous applications, although, it makes some major issues arise, like the necessity to deal with a high-dimensional space of parameters, or the presence of hardware constraints to be respected while exploring policies, in order to avoid any serious damage. These problems in some cases can have a critical influence on the success of the reinforcement learning. In particular, when learning tasks with two robotic manipulators, the amount of parameters in the motion representation becomes too large to obtain a good solution with a limited amount of executions.

Summary

The learning of bimanual robotic tasks, i.e., tasks executed by two manipulators together, can be particularly important in the new scenarios opened by the rise of humanoid robotics, one of the most interesting trend currently in the field. The work presented wants to build a method to simplify the dimensionality of parameter space in this particular context, exploiting the presence of symmetries between the movements executed by the two arms. The aim is to develop a reduced-order representation of the bimanual motion, with the purpose of increase the speed of learning process. In chapter 1, kinematics of the used robots is studied, in order to know how to correctly command the position of the robots while executing a task. Robotic movements are then modeled using *Probabilistic Movement Primitives* (ProMPs), a stochastic interpretation of robot movements (details in chapter 2). The first objective is to develop a symmetrization method for those kind of policies, and this part is treated in chapter 3. This will give the chance of representing the movement of two robotic arms, with only a single ProMP (instead of two, one for each arm), from which obtain the second policy applying symmetrization. In this way the amount of parameters representing motion can be halved. The most common kind of symmetry is the one defined by a plane, but also other cases can be explored, e.g., spherical or cylindrical symmetry. If the symmetry surface is not explicitly given in the bimanual task description, it is critical to have a reliable method to estimate it in order to exploit it in the learning process. In chapter 4 it is reported a way to obtain this estimation of the parameters describing the symmetry surface from the initially demonstrated trajectories. Finally, in chapter 5 it is defined a symmetric policy representation for bimanual task, that depends only on a single ProMP and a symmetry surface. The effectiveness of this parameter reduction has been tested applying it in reinforcement learning of some tasks, in comparison to the results obtained by the standard way of proceeding, that model the bimanual task with two separated ProMPs, one for each robotic arm.

1

Robot Positioning

The first problem that needs to be faced is the control of the position of a robot, otherwise it would be not possible to carry out affectively the reinforcement learning process. Serial robots are composed of concatenated joints, forming an open chain structure, it is then necessary to have reliable methods to get end-effector position from joint angles, and vice-versa. This chapter provides descriptions of forward and inverse kinematics for robot manipulators, together with an analysis of reference frames on which movements can be expressed, issue that needs to be addressed when using two different robots.

1.1 Forward and inverse kinematics

It is considered a serial robot with *m* joints, and let $W \subseteq \mathbb{R}^n$ be its *workspace* (reachable positions) in a space of dimension *n* ($n = 6$ for spatial position and orientation, as in the case of robotic manipulators), and $Q \subseteq \mathbb{R}^m$ be its *joint-space*, the space of feasible joint positions. Then it is possible to define:

- **The Forward Kinematics** as the function mapping a joint position in the joint-space Q into a end-effector position in the workspace W, by construction $W = Im_Q(f)$.

  $$f : Q \subseteq \mathbb{R}^m \rightarrow W \subseteq \mathbb{R}^n$$

  $$\boldsymbol{q} \mapsto \boldsymbol{x}$$

  The forward kinematics function for serial manipulators can be obtained easily with *Denavit-Hartenberg* (D-H) coefficients [4] (see section 1.2).

- **The Inverse Kinematics** as the function that maps an end-effector position in the workspace W into a position in the joint-space Q:

  $$h : W \subseteq \mathbb{R}^n \rightarrow Q \subseteq \mathbb{R}^m$$

  $$\boldsymbol{x} \mapsto \boldsymbol{q}$$

  In serial robots *h* can have multiple solutions for a single $\boldsymbol{x}$, or even infinite solutions for $m > n$ or in a *degenerate position*. Obtaining a closed expression for *h* can be very complicated or even impossible.

1.2 Denavit-Hartenberg parameters

When considering a serial open-chain manipulator, a systematic way of expressing its forward kinematics is using the *Denevit-Hartenberg* convention [4]. This consists in defining the relative position and orientation of two consecutive links, and determining a transformation matrix between them, by performing the following operations (see Fig. 1.1):

- Set axis z_i as the axis of rotation or movement oj joint *i*+1.
- Find the common normal line s between z_{i-1} and z_i. Define the two points of intersection $O_i = s \bigcap z_i$ and $O'_{i-1} = s \bigcap z_{i-1}$, O_i is the origin of frame *i*.
- Define the axis x_i in the direction of *s*, from joint *i* to joint *i*+1.
- $y_i = z_i \times x_i$, defined following the right-hand convention.

Define now

- a_i is the distance (with sign) from O'_i to O_i along axis x_i.
- d_i is the distance (with sign) from O_{i-1} to O'_i along axis z_{i-1}.
- α_i is the angle (with sign following the right-hand convention) from axis z_{i-1} to z_i around x_i.
- θ_i is the angle (with sign following the right-hand convention) from axis x_{i-1} to x_i around z_{i-1}.

Parameters α_i and a_i are always constant, while θ_i is a variable if joint *i* is rotational and d_i is a variable if the joint *i* is prismatic.

Furthermore, some ambiguous situations must be taken into account:

- The first frame has only z_0 direction defined, so O_0 and x_0 can be chosen arbitrarily.
- For the last frame (*n*), as no joint *n+1* exists, z_n is not uniquely defined and can also be chosen arbitrarily.
- When two consecutive axes intersect, x_i is arbitrarily chosen.
- When two consecutive axes are parallel, z_i can be placed along a set of parallel lines.
- When joint *i* is prismatic, then the parameter *d* calculated is taken as an offset on the variable d_i.
- When joint *i* is rotational, then the parameter θ calculated is taken as an offset on the variable θ_i.

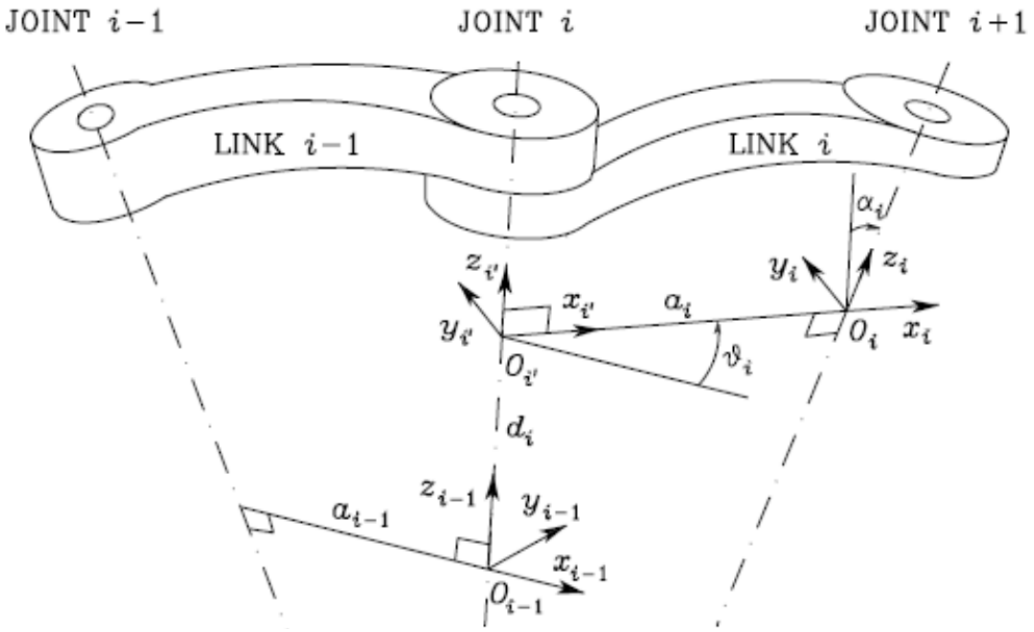


Figure 1.1: *D-H parameters schematic definition (image taken from [5]).*

With these parameter and considerations, the homogeneous transformation from frame *i-1* to frame *i* is given by

$$ {}^{i-1}T_i = \begin{bmatrix} cos(\theta_i) & -sin(\theta_i)cos(\alpha_i) & sin(\theta_i)sin(\alpha_i) & a_i cos(\theta_i) \\ sin(\theta_i) & cos(\theta_i)cos(\alpha_i) & -cos(\theta_i)sin(\alpha_i) & a_i sin(\theta_i) \\ 0 & sin(\alpha_i) & cos(\alpha_i) & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{1.1} $$

and for several joints, the homogeneous transformation from base frame to the end-effector frame is given by the concatenation ${}^bT_{ee} = {}^b\ T_0\ {}^0T_1 \cdots\ {}^{m-1}T_m\ {}^mT_{ee}$. ${}^bT_0$ is the homogeneous transformation between a base frame and the joint 1 frame, ${}^mT_{ee}$ is the transformation giving end-effector relative position with respect to the end joint. ${}^bT_{ee}$ represents the end-effector position in base frame given joint positions, i.e., the forward kinematics for a serial robot manipulator.

1.3 Solution of inverse kinematics

For a serial robot, forward kinematics equations are often easy to obtain (for example, using D-H parameters), hence, a possible approach to inverse kinematics computation, is to obtain the expression of inverses of those equations. As many trigonometric functions appear when dealing with rotational joints, this inversion can require some algebraic skills, as it will be non-trivial, or even impossible. Moreover, multiple solutions of trigonometric functions must be considered. The complexity of the procedure grows with the complexity of the robot geometry, and it is very useful to split the problem into several smaller ones when possible. This approach proves itself very useful in treating the inverse kinematics problem in the case of *Barrett WAM Arm*, the robotic manipulator adopted in this work (Fig. 1.2).

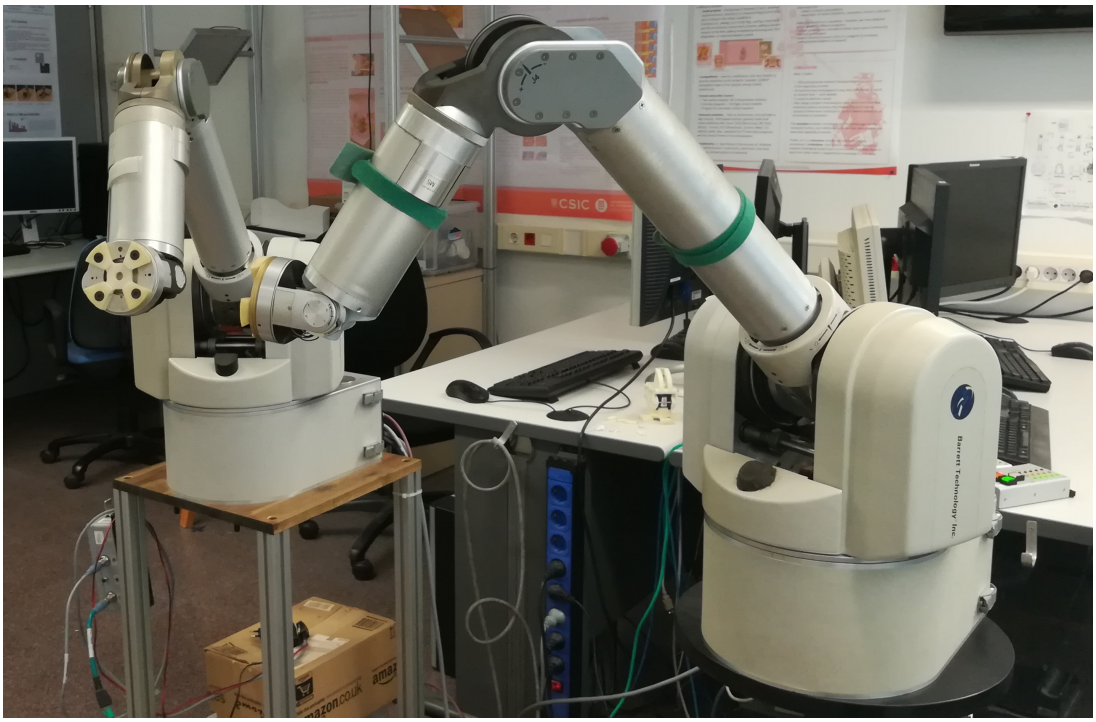


Figure 1.2: *The two Barrett WAM Arms used.*

1.3.1 Barrett WAM Arm

Barrett WAM Arm is a highly dexterous backdrivable robot manipulator with human-like kinematics [6]. The model adopted is characterized by 7 DoF and it is a very common choice for applications in the field of collaborative robotics. D-H parameters description of the robot manipulator are given in Table 1.1 (values taken from support manual [7]).

Table 1.1: *7 DoF WAM arm D-H parameters.*

link i	a_i	α_i	d_i	θ_i	θ_i^{min}	θ_i^{max}
1	0	$-\pi/2$	0	θ_1	-2.6	2.6
2	0	$\pi/2$	0	θ_2	-2.0	2.0
3	0.045	$-\pi/2$	0.55	θ_3	-2.8	2.8
4	-0.045	$\pi/2$	0	θ_4	-0.9	3.1
5	0	$-\pi/2$	0.3	θ_5	-4.8	1.3
6	0	$\pi/2$	0	θ_6	-1.6	1.6
7	0	0	0.06	θ_7	-2.2	2.2

Now, for solving its inverse kinematics, it has to be kept in mind that this robot has one redundant degree of freedom. What can be done is to solve the system with one fixed angle, and then perform some optimization to find the best value (as described in [8]).

In this robot, the last 3 DoF consist of a so-called *spherical wrist*, which can be treated later independently from the rest of the joints (see section 1.3.2). In fact, given the desired homogeneous transformation describing position (P_l) and orientation $\left(R_l := \left[\boldsymbol{n}|\boldsymbol{s}|\boldsymbol{a}\right]\right)$ of the last joint frame with respect to the

fourth:

$$T_l = \begin{bmatrix} \boldsymbol{n} & \boldsymbol{s} & \boldsymbol{a} & P_l \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{1.2}$$

the inverse kinematics of the spherical wrist center $P_w = P_l - d_7 \cdot \boldsymbol{a}$ can be considered. This reduces the 7 DoF positioning problem to a 4 DoF problem, where the task is to reach point P_w (of dimension 3) without paying attention to the orientation of end-effector.

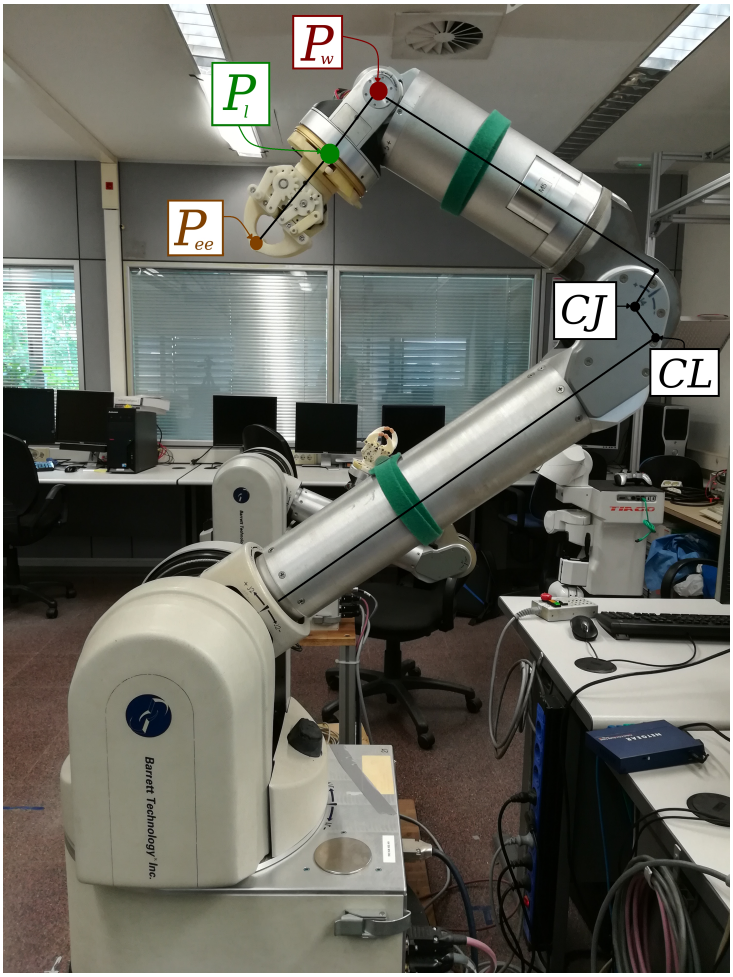


Figure 1.3: *Representation of P_w, P_l and P_{ee}. Points CL and CJ are used in the optimization of the redundant degree of freedom.*

Thus, the problem is now defined as a 4 DoF robot with a three-dimensional task, so it has to be fixed one of the four angle to calculate a first solution of the inverse kinematics. The best joint to be fixed is the first one, mainly because it moves all the inertia of the robot, and minimizing its angle variation the total energy spent during a movement is reduced.

Firstly, angle θ_4 can be computed from the equation of distance from the origin to the spherical wrist center point (consider $a := a_3 = -a_4 = 0.045$ and ${}^4P_w = [0,\ 0,\ d_5,\ 1]^T = [0,\ 0,\ 0.3,\ 1]^T$):

$$||{}^0P_w||^2 = ||{}^0T_1(\theta_1)\ {}^1T_2(\theta_2)\ {}^2T_3(\theta_3)\ {}^3T_4(\theta_4)\ {}^4P_w||^2 = \dots \tag{1.3}$$

$$= d_3^2 + d_5^2 + d_7^2 + 2(d_5 \cdot a + d_3 \cdot a)sin(\theta_4) + 2(d_3 \cdot d_5 + a^2)cos(\theta_4)$$

which can be solved to obtain two possible values of θ_4, called *inner elbow* and *outer elbow* configurations, depending on the sign of θ_4. Then, from previous expression, also equality (1.4) holds (the notation $c_i := cos(\theta_i)$ and $s_i := sin(\theta_i)$ has been adopted).

$$ {}^2T_3(\theta_3)\ {}^3T_4(\theta_4)\ {}^4P_w = ({}^1T_2(\theta_2))^{-1}\ ({}^0T_1(\theta_1))\ {}^0P_w\ : \tag{1.4} $$

$$ \begin{bmatrix} 0.3c_3s_4 - \frac{9}{200}c_3c_4 + \frac{9}{200}c_3 \\ 0.3s_3s_4 - \frac{9}{200}s_3c_4 + \frac{9}{200}s_3 \\ \frac{11}{20} + 0.3c_4 + \frac{9}{200}s_4 \\ 1 \end{bmatrix} = \begin{bmatrix} s_1c_2(P_w^0)_x + s_1c_2(P_w^0)_y - s_2(P_w^0)_z \\ -s_1(P_w^0)_x + c_1(P_w^0)_y \\ c_1s_2(P_w^0)_x + s_1s_2(P_w^0)_y + c_2(P_w^0)_z \\ 1 \end{bmatrix} $$

from where, knowing θ_1 and θ_4, two solutions for θ_2 can be found solving:

$$ \frac{11}{20} + 0.3c_4 + \frac{9}{200}s_4 = c_1s_2(P_w^0)_x + s_1s_2(P_w^0)_y + c_2(P_w^0)_z \tag{1.5} $$

and, with θ_2, $c_3 = cos(\theta_3)$ and $s_3 = sin(\theta_3)$ are obtained from:

$$ 0.3c_3s_4 - \frac{9}{200}c_3c_4 + \frac{9}{200}c_3 = s_1c_2(P_w^0)_x + s_1c_2(P_w^0)_y - s_2(P_w^0)_z \tag{1.6} $$

$$ 0.3s_3s_4 - \frac{9}{200}s_3c_4 + \frac{9}{200}s_3 = -s_1(P_w^0)_x + c_1(P_w^0)_y \tag{1.7} $$

To finally get $\theta_3 = atan2(s_3, c_3)$. The last angles $(\theta_5, \theta_6, \theta_7)$, describing position of spherical wrist, can be obtained separately, as shown in next section.

1.3.2 Spherical wrist

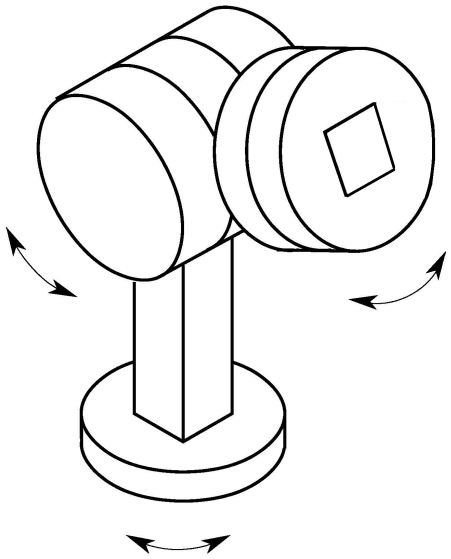


Figure 1.4: *Schematic representation of a spherical wrist.*

A typical architecture for spatial robotic arms is to end in a *spherical wrist* (see Fig. 1.4). Its D-H parameters characterization is given by Table 1.2.

Table 1.2: *Spherical wrist D-H parameters.*

link *i*	a_i	α_i	d_i	θ_i
i	0	$-\pi/2$	d_i	θ_i
i+1	0	$\pi/2$	0	θ_{i+1}
i+2	0	0	d_{i+2}	θ_{i+2}

The forward kinematics can hence be expressed as in (1.8) where $c_i := cos(\theta_i)$ and $s_i := sin(\theta_i)$.

$$ {}^{i-1}T_{i+2} = {}^{i-1}T_i \, {}^{i}T_{i+1} \, {}^{i+1}T_{i+2} = \begin{bmatrix} c_i c_{i+1} c_{i+2} - s_i s_{i+2} & -c_i c_{i+1} s_{i+1} - s_i c_{i+2} & c_i s_{i+1} & c_i s_{i+1} d_{i+2} \\ s_i c_{i+1} c_{i+2} - c_i s_{i+2} & -s_i c_{i+1} s_{i+1} - c_i c_{i+2} & s_i s_{i+1} & s_i s_{i+1} d_{i+2} \\ -s_{i+1} c_{i+2} & s_{i+1} s_{i+2} & c_{i+1} & c_{i+1} d_{i+2} \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{1.8} $$

With this transformation, supposing a robot of *m* DoF, the last three of them correspond to a spherical wrist. Thus, in case of *Barrett WAM Arm*, $i = 5$ and the links defining the spherical wrist are the fifth, the sixth and the seventh.

Given an objective homogeneous transformation for the end-effector (with respect to the base frame) ${}^0T_{ee}$, this can be expressed as ${}^0T_{ee} = {}^0T_{m-3} \, {}^{m-3}T_m \, {}^mT_{ee}$, from where it can be defined $T_l := {}^{m-3}T_m$, the spherical wrist homogeneous transformation. Then: ${}^0T_m = {}^0T_{m-3} \, T_l \Longrightarrow T_l = ({}^0T_{m-3})^{-1} \, {}^0T_m$.

So, in the case of *Barret WAM Arm*, after computing the inverse kinematics for the wrist position P_w (as described in previous section), and given the desired $R_l := \Big[\boldsymbol{n}|\boldsymbol{s}|\boldsymbol{a}\Big]$, solutions for the final joints of the spherical wrist are:

$$\theta_5 = atan2(\epsilon a_y, \epsilon a_x) \tag{1.9}$$

$$\theta_6 = atan2(\epsilon\sqrt{a_x^2 + a_y^2}, a_z) \tag{1.10}$$

$$\theta_7 = atan2(\epsilon s_z, -\epsilon n_z) \tag{1.11}$$

where $\epsilon = \pm 1$, having then two sets of solutions, and $\boldsymbol{n} = [n_x, n_y, n_z]$, $\boldsymbol{s} = [s_x, s_y, s_z]$, $\boldsymbol{a} = [a_x, a_y, a_z]$.

1.3.3 Redundant degree of freedom optimization

The set of equations derived before (1.3-1.5-1.6-1.7-1.9-1.10-1.11) find, if possible, the joint positions of the WAM arm verifying $\theta_1 = \theta_1^0$, fixed value for the first joint. If no solution exists, another value of θ_1^0 is tried until a solution is found. Usually this process starts from a common reference value and then making bigger and bigger changes to it.

To perform an optimization of the joints' positions of the robot, knowing a first solution of its kinematics, [9] and [10] propose that, for this kind of robots their redundant degree of freedom comes from the elbow rotating around the axis going from the base of the robot to its wrist, that can de defined by vector $u_w =< P_w - P_0 >$. To do such rotation, points *CL,CJ* are considered (as defined in Fig. 1.3) and rotated of an angle ϕ around such axis, with the equations:

$$C_\phi = R_\phi C, \quad \text{where} \quad C \text{ indicates } CL, \text{ or } CJ \tag{1.12}$$

$$R_\phi = I_3 + sin(\phi)[u_w\times] + (1 - cos(\phi))[u_w\times]^2, \tag{1.13}$$

being $[u_w\times]$ the skew-symmetric matrix of vector u_w. Note that u_w is an eigenvector of eigenvalue 1 of R_ϕ: $R_\phi \cdot u_w = u_w + sin(\phi)[u_w\times] \cdot u_w + (1 - cos(\phi))[u_w\times]^2 \cdot u_w = u_w$, as $[u_w\times] \cdot u_w = 0$. So, from R_ϕ, the new rotated points CL_ϕ and CJ_ϕ can be computed with (1.12). The new angles are $\theta_1 = atan2((CL_\phi)_y, (CL_\phi)_x)$ $\theta_2 = acos((CL_\phi)_z/d_3)$ and θ_3 can be obtained solving the system:

$$\frac{CJ_\phi - CL_\phi}{a} = \begin{bmatrix} c_1c_2 & -s_1 \\ s_1c_2 & c_1 \\ {}_2 & 0 \end{bmatrix} \cdot \begin{bmatrix} c_3 \\ s_3 \end{bmatrix} \tag{1.14}$$

where trigonometric functions are indicated applying the usual convention ($c_i = cos(\theta_i)$, $s_i = sin(\theta_i)$). Since the position of wrist does not change with this kind of rotation, angle θ_4 will be the same if the *elbow configuration* (sign of θ_4) is constant. From the first 4 angles, associated θ_5, θ_6 and θ_7 can be obtained solving again the spherical wrist equations. Note that, if $\boldsymbol{\theta_{sol}} = [\theta_1, \theta_2, \theta_3, \theta_4, \theta_5, \theta_6, \theta_7]$ is a solution of the inverse kinematics, so it is $[\theta_1 \pm \pi, -\theta_2, \theta_3 \pm \pi, \theta_4, \theta_5 \pm \pi, -\theta_6, \theta_7 \pm \pi]$; thus, for each solution on θ_4, there are 4 solutions of the inverse kinematics. It is recommended to use the one closer to a reference position (or the previous position).

The final solution of the inverse kinematics will be the result of an optimization depending on the solution rotation angle ϕ. To do so, it is decided to minimize a weighted norm of the difference between solution vector $\boldsymbol{\theta_{sol}}$ and the positions of joints in previous time step $\boldsymbol{\theta_{init}}$ (when these positions are not available, $\boldsymbol{\theta_{init}}$ is considered a vector of zeros): $(\boldsymbol{\theta_{sol}} - \boldsymbol{\theta_{init}})^T \cdot W \cdot (\boldsymbol{\theta_{sol}} - \boldsymbol{\theta_{init}})$. W may vary on the needs of the user, depending on the task being performed, in this work it is chosen $W = \text{diag}(2.5, 2.0, 2.0, 1.5, 1.0, 1.0, 1.0)$. Weights have been empirically tuned after several trials, trying to imitate a human-like movement of the arm. Note that the first solution of the inverse kinematics, obtained analytically, can be associated to the weight matrix $W = \text{diag}(1, 0, 0, 0, 0, 0, 0)$.

1.4 Reference frames for robot trajectories

Movements describing the bimanual task are initially demonstrated moving the robots held in a gravity compensation state, recording joint positions for the two arms. After computing the forward kinematics, each end-effector trajectory is obtained with respect to its own base frame, so it is necessary to express the two movements in the same reference to compare them. This common frame can be the base frame of one of the two robot-arm; let's call it first arm from now on.

The relative position and orientation between the two robot bases can be expressed using a transformation matrix ${}^{b_1}T_{b_2}$. This same matrix define the roto-translation needed to change from trajectory points from one reference frame to the other. Rotation is described by a 3×3 matrix ${}^{b_1}R_{b_2}$, that denotes the orientation in space of the the second frame with respect to the first, and a translation vector ${}^{b_1}L_{b_2}$ describing the origin of the second frame with respect to the first. In (1.15) it is shown precisely how to build ${}^{b_1}T_{b_2}$ and the procedure followed to perform the frame change.

$$\begin{bmatrix} {}^{b_1}x_2 \\ {}^{b_1}y_2 \\ {}^{b_1}z_2 \\ 1 \end{bmatrix} = {}^{b_1}T_{b_2} \cdot \begin{bmatrix} {}^{b_2}x_2 \\ {}^{b_2}y_2 \\ {}^{b_2}z_2 \\ 1 \end{bmatrix} \qquad \text{with} \qquad {}^{b_1}T_{b_2} = \begin{bmatrix} {}^{b_1}R_{b_2} & {}^{b_1}L_{b_2} \\ \mathbf{0}_{1\times 3} & 1 \end{bmatrix} \tag{1.15}$$

All along this work, it has been chosen to define movement policies of each robot-arm in its own base frame. This has to be taken into account when working with symmetries during the learning process, changing reference frames in order to work with the two trajectories in space.

Kinematics of the *Barrett WAM Arm* has been modeled with MATLAB and the *Robotics Toolbox* [11] to obtain an immediate way of computing its forward kinematics from recorded positions of joints, getting the demonstrated trajectories of the two end-effectors. For sake of simplicity, it has been made the decision to fix the spherical wrist position aligned with the fourth link ($\theta_5 = \theta_6 = \theta_7 = 0$). This allows to define movement policies of the robot directly as trajectories of the end-effector in three-dimensional space, without the necessity of considering the orientation of spherical wrist. On the other hand, when a movement in 3D space is given by a policy, associated joint trajectories can be obtained from the inverse kinematics, and passed as commands to be executed by the robots.

In the next chapter Probabilistic Movement Primitives are introduced, as the representation chosen for the movements of *Barrett WAM Arms*. These movement primitives are used to model the trajectories of robots' end-effectors in space.

2

Probabilistic Movement Primitives

Movement Primitives (MPs) are a well-established approach for representing and learning basic movements in robotics. MP formulation gives a compact representation of the robot's policy that allow to deal with the inherently continuous robot movements. They can be used in imitation and reinforcement learning, as well as adapting to different scenarios, modulating accordingly their parameters. Over the last years, *Dynamic Movement Primitives* [12] (DMPs) have been widely used for motion representation and learning. However, DMPs are a deterministic approach to motion representation, thus they are not capable of representing motion variability. In contrast, the recently proposed *Probabilistic Movement Primitives* [13] (ProMPs) approach is capable of capturing the variance of a set of demonstrations to a robot of the same task, and then reproducing the trajectory with the same variance over time.

2.1 Probablistic trajectory representation

A single movement execution is modeled as a trajectory $\boldsymbol{\tau} = \{q_t\}_{t=0...T}$ defined by the position of the robot over time. In the framework considered a MP describes multiple ways to execute a movement, leading to a probability distribution over trajectories. This representation allows to capture multiple demonstrations with high-variability, being able to model the time-varying variance of the trajectories.

ProMPs use a weight vector $\boldsymbol{w}$ to compactly represent a single movement. The probability of observing a certain trajectory $\boldsymbol{\tau}$ given the underlying weight vector $\boldsymbol{w}$ is given as a linear basis function model.

$$q_t = \Phi_t^T \boldsymbol{w} + \epsilon_q, \qquad p(\boldsymbol{\tau}|\boldsymbol{w}) = \prod_t \mathcal{N}(q_t|\Phi_t^T \boldsymbol{w}, \Sigma_q), \tag{2.1}$$

where Φ_t defines a time-dependent basis matrix, composed by a sequence of Gaussian functions $b_i(t) = e^{-h\,(t-c_i)^2}$ over time, $\epsilon_q \sim \mathcal{N}(0, \Sigma_q)$ is a zero-mean Gaussian observation noise, and $\boldsymbol{w}$ a weight vector (one weight associated to one basis function). By weighting the basis functions Φ_t with the parameter vector $\boldsymbol{w}$, it is possible to represent the mean of a trajectory.

The structure of the basis functions must be finely tuned with respect to movement they need to represent. A common choice is to distribute uniformly the Gaussian functions' centers c_i over time, and

to set their numbers M and their amplitude (defined by the value of h) in such a way that the time interval is well-covered at each instant (Fig. 2.1).

In order to capture the variance of the trajectories, a p.d.f. $p(\boldsymbol{w};\boldsymbol{\theta})$ over the weight vector $\boldsymbol{w}$, with parameters $\boldsymbol{\theta}$, is introduced. The distribution of the trajectory $p(\boldsymbol{\tau};\boldsymbol{\theta})$ can be obtained by marginalizing out $\boldsymbol{w}$: $p(\boldsymbol{\tau};\boldsymbol{\theta}) = \prod_t p(q_t;\boldsymbol{\theta}) = \prod_t \int p(q_t|\boldsymbol{w})p(\boldsymbol{w};\boldsymbol{\theta})d\boldsymbol{w}$. In this work it is assumed that the observation noise ϵ_q can be ignored being possible to model any movement well enough with a weighted sum of basis functions, i.e., $q_t = \Phi_t^T \boldsymbol{w}$. Consequently, the trajectory distribution depends only on the parameters $\boldsymbol{\theta}$ of $p(\boldsymbol{w};\boldsymbol{\theta})$.

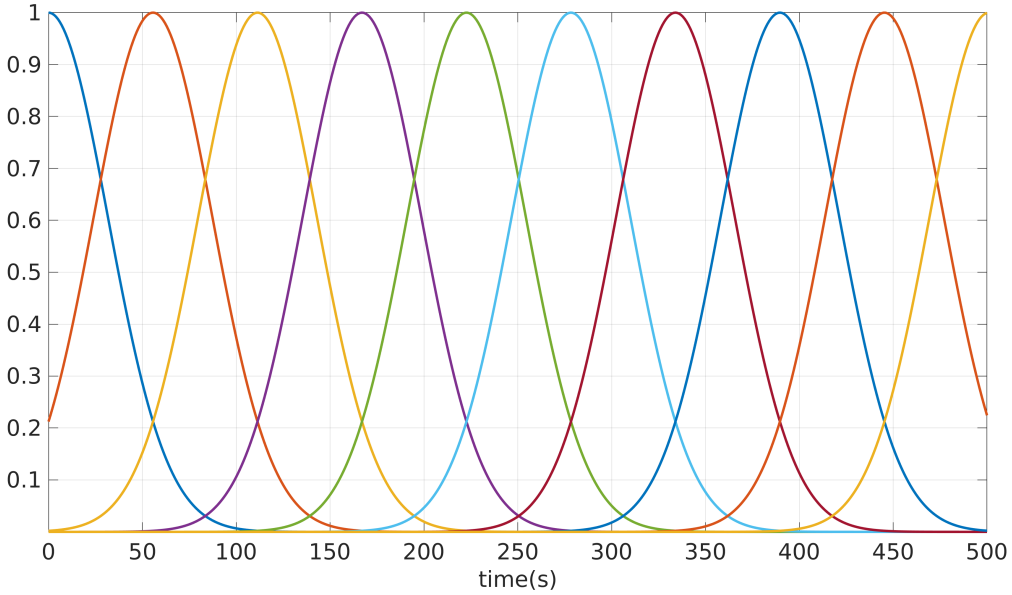


Figure 2.1: *Example of Gaussian basis functions (M=10), equally distributed time (T=500s).*

2.2 Learning from demonstrations

It is fundamental for a MP representation to have the possibility to acquire easily the parameters of a single primitive from demonstrations. It is assumed a Gaussian distribution $p(\boldsymbol{w};\boldsymbol{\theta}) = \mathcal{N}(\mu_{\boldsymbol{w}}, \Sigma_{\boldsymbol{w}})$ over the weight vector $\boldsymbol{w}$, i.e., $\boldsymbol{\theta} = [\mu_{\boldsymbol{w}}, \Sigma_{\boldsymbol{w}}]$. Having neglected the effects of observation noise ϵ_q, the distribution over state $p(q_t|\boldsymbol{\theta})$ for time instant t is given by

$$p(q_t|\boldsymbol{\theta}) = p(q_t|\mu_{\boldsymbol{w}}, \Sigma_{\boldsymbol{w}}) = \mathcal{N}(q_t|\Phi_t^T \mu_{\boldsymbol{w}}, \Phi_t^T \Sigma_{\boldsymbol{w}} \Phi_t) \tag{2.2}$$

and, thus, is possible to evaluate the mean and the variance of the trajectory at each time t.

The parameters $\mu_{\boldsymbol{w}}$ and $\Sigma_{\boldsymbol{w}}$ can be learned from multiple demonstrations of the desired trajectories: given K_d demonstrations $\boldsymbol{\tau}_k$, the ProMP representation of each trajectory in matrix form is given in 2.3 (time interval $[0 \dots T]$ has been sampled in N_t equally-spaced time steps).

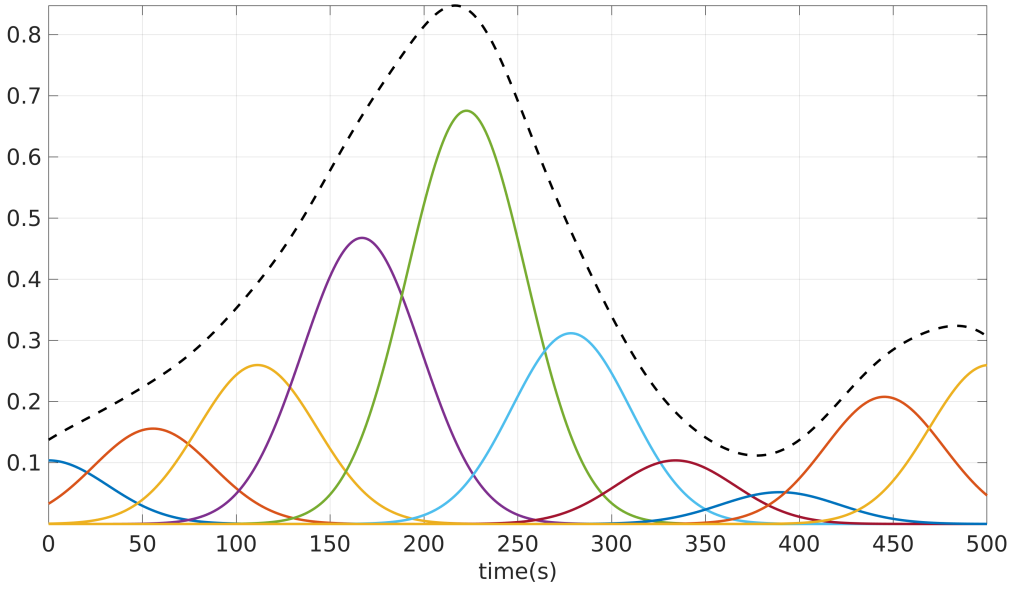


Figure 2.2: *Weighted sum of Gaussian basis functions* $\Phi_t^T \boldsymbol{w}$ *modelling a trajectory* q_t.

$$\begin{bmatrix} q_1 \\ \vdots \\ q_{N_t} \end{bmatrix}^k = \Psi^T \boldsymbol{w}^k \quad \text{with } \Psi^T := \begin{bmatrix} \Phi_1^T \\ \vdots \\ \Phi_{N_t}^T \end{bmatrix} \quad \text{for } k = 1 \dots K_d \tag{2.3}$$

From (2.3) the weight vector $\boldsymbol{w}^k$ associated to each demonstrations can be computed applying the *Moore-Penrose pseudoinverse* in expression (2.3):

$$\boldsymbol{w}^k = \left(\Psi^T\right)^+ \begin{bmatrix} q_1 \\ \vdots \\ q_{N_t} \end{bmatrix}^k \qquad \text{for } k = 1 \dots K_d \tag{2.4}$$

Having the weight vectors $\boldsymbol{w}^k$ derived from demonstrations it is possible to compute the associated mean and covariance matrix describing the p.d.f. $\boldsymbol{w} \sim \mathcal{N}(\mu_{\boldsymbol{w}}, \Sigma_{\boldsymbol{w}})$ using Unbiased Maximum Likelihood Estimation (UMLE) (2.5-2.6).

$$\mu_{\boldsymbol{w}} = \frac{1}{K_d} \sum_{k=1}^{K_d} \boldsymbol{w}^k \tag{2.5}$$

$$\Sigma_{\boldsymbol{w}} = \frac{1}{K_d - 1} \sum_{k=1}^{K_d} (\boldsymbol{w}^k - \mu_w)^T (\boldsymbol{w}^k - \mu_w) \tag{2.6}$$

In some cases it may be necessary to increase the variance in $\Sigma_{\boldsymbol{w}}$ because the demonstrations are not enough to guarantee that the covariance matrix found is not singular. To do so a constant value its added to diagonal entries, this value is generally chosen to be inferior to the smallest not-null eigenvalue of the original matrix, to avoid excessive deformation of the covariance.

After this process it is obtained a first *ProMP* representation of the movement. In order to get an actual trajectory $\{\bar{q}_t\}_{t=0...T}$ (a so-called rollout) from the MP, a weight vector $\bar{\boldsymbol{w}}$ need to be sampled from the p.d.f. and multiplied for the basis function matrix, obtaining $\bar{q}_t = \Phi_t^T \bar{\boldsymbol{w}}$, for each time instant.

ProMP representation can be extended to the case of a multivariate trajectory of general dimension *D*. Each degree of freedom is assigned to *M* weights $\boldsymbol{w}_i$ that can be combined in a single vector associated to a block basis function matrix. Hence, the weights p.d.f. is a multivariate Gaussian distribution of dimension $D \cdot M$. In this way, it possible to capture correlations between distinct degree of freedoms through the covariance between different weights vector $\boldsymbol{w}_i$. In the context of this work, ProMPs are used to represent and learn trajectories in the three-dimensional space, $\boldsymbol{\tau} = \left\{[x_t,\ y_t,\ z_t]^T\right\}_{t=0...T}$ (example in Fig. 2.3), thus, it has been considered a weight vector of dimension $3 \cdot M$ (*M* weights for each direction *X*, *Y* and *Z*). So, the position at time instant *t* is given by:

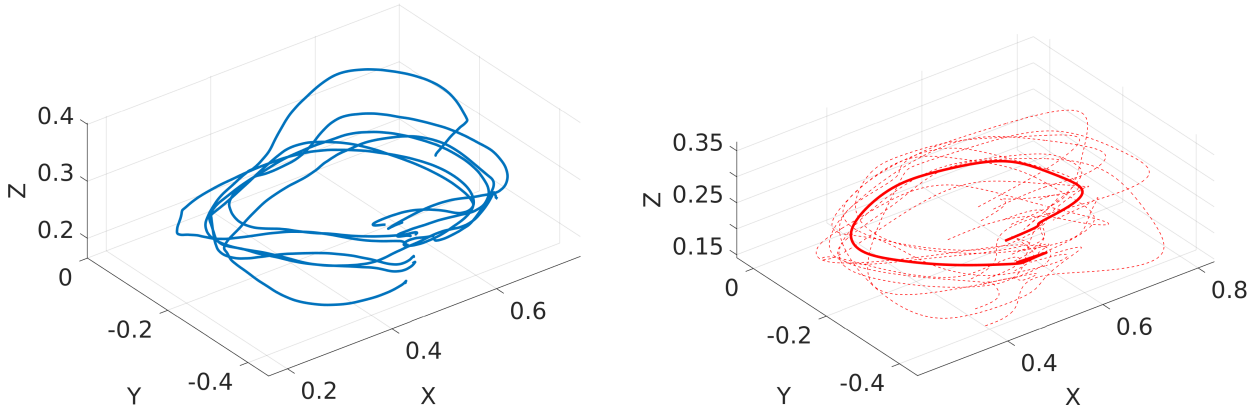
$$\begin{bmatrix} x_t \\ y_t \\ z_t \end{bmatrix} = \boldsymbol{\Phi}_t^T \begin{bmatrix} \boldsymbol{w}_x \\ \boldsymbol{w}_y \\ \boldsymbol{w}_z \end{bmatrix} \quad \text{with} \quad \boldsymbol{\Phi}_t^T := \begin{bmatrix} \Phi_t & 0 & 0 \\ 0 & \Phi_t & 0 \\ 0 & 0 & \Phi_t \end{bmatrix}^T \tag{2.7}$$

The parameters defining the Gaussian distribution for the combined weight vector can be obtained extending the expressions given before in the one-dimensional case (2.3-2.4). Calling $\mathbf{x} := [x_1 \dots x_{N_t}]^T$, $\mathbf{y} := [y_1 \dots y_{N_t}]^T$ and $\mathbf{z} := [z_1 \dots z_{N_t}]^T$, the resulting expressions are:

$$\begin{bmatrix} \mathbf{x} \\ \mathbf{y} \\ \mathbf{z} \end{bmatrix}^k = \boldsymbol{\Psi}^T \begin{bmatrix} \boldsymbol{w}_x \\ \boldsymbol{w}_y \\ \boldsymbol{w}_z \end{bmatrix}^k \quad \text{with} \quad \boldsymbol{\Psi}^T := \begin{bmatrix} \Psi & 0 & 0 \\ 0 & \Psi & 0 \\ 0 & 0 & \Psi \end{bmatrix}^T \quad \text{for } k\text{=}1 \dots K_d \tag{2.8}$$

$$\begin{bmatrix} \boldsymbol{w}_x \\ \boldsymbol{w}_y \\ \boldsymbol{w}_z \end{bmatrix}^k = (\boldsymbol{\Psi}^T)^+ \begin{bmatrix} \mathbf{x} \\ \mathbf{y} \\ \mathbf{z} \end{bmatrix}^k \quad \text{for } k\text{=}1 \dots K_d \tag{2.9}$$

Finally, mean and covariance of the combined weight vector can be computed using UMLE, exactly as in the one-dimensional case (2.5- 2.6).



(a) *Demonstrated trajectories* (b) *Learned ProMP's rollouts*

Figure 2.3: *Example of learning from demonstrations with ProMPs applied to a movement in the 3D pace. The starting demonstrations are shown in (a), while in (b) the mean trajectory (thick line) and some rollouts sampled from the resulting ProMP (dashed lines) are reported.*

A ProMP is associated to each robotic-arm describing the movements to be executed when performing a bimanual task. If a symmetry is present between the movements of two robots, it can be used to describe both of them: starting from a single ProMP, representing movement for one arm, it can be found its symmetric ProMP for the second.

3

Symmetrization Methods

The presence of a symmetry between the movement of two robotic arms can be exploited to reduce the dimensionality of the bimanual task policy representation: the ProMP of the second arm can be expressed as a symmetrization of the ProMP of the first one, in this way the bimanual task policy is given by only a single MP and a surface, reducing considerably the number of parameters. The most common type of symmetry is defined by a plane, but this is not the only possible situation, and the cases of spherical and cylindrical surfaces are also taken into account. In this chapter it is firstly found a method of symmetrization for Gaussian probability distributions, to be used in second place as a base to develop a symmetrization methods for ProMPs.

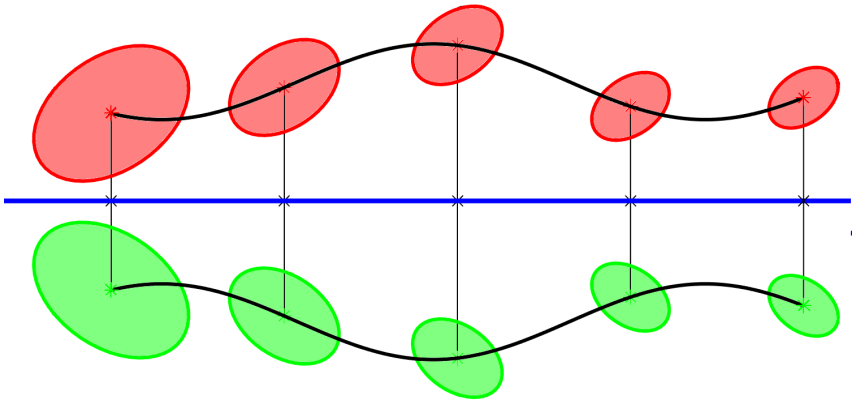


Figure 3.1: *Symmetric trajectories as time-dependent multivariate Gaussian probability density functions in a simplified 2D example. Gaussians have been represented as ellipses of uncertainty.*

3.1 Gaussian distributions symmetrization

Being a ProMP a representation of the robot movements that describes a trajectory as a time-dependent multivariate Gaussian probability density function, first of all, it is necessary to find a method to symmetrize a distribution $\mathcal{N}(\mu, \Sigma)$ with respect to a given surface (a plane, a sphere or a cylinder). This will be the base on which the generalized ProMP symmetrization method will be built upon: from the symmetric mean and covariance matrices at each time step, it will be possible to "fit" a symmetric ProMP. Being

the trajectories considered a sequence of points in the 3D space, the symmetrization methods developed are thought to be used with multivariate Gaussian distributions of dimension 3, but they can be adapted straightforwardly to spaces of different dimensions.

3.1.1 Planar symmetries

A Gaussian p.d.f. is characterized by two elements, its expected value and its covariance matrix. The first element to be symmetrized is the mean value $\mu = [x_\mu, y_\mu, z_\mu]^T$. This is a simple geometric problem that consists in finding the symmetric of a point in the space with respect to a plane $\pi : ax + by + cz + d = 0$. The direction perpendicular to the plane is defined by the vector $< [a, b, c]^T >$, so the line perpendicular to π going through the point μ can be expressed in parametric form as $\mu + t \cdot < [a, b, c]^T >$ (with t varying parameter). To obtain the symmetric point, it is necessary to find the projection of μ on the plane. The solution of system (3.1) is the intersection between the perpendicular line and the symmetry plane, i.e., the projected point needed.

$$\begin{cases} x = x_\mu + a\ t \\ y = y_\mu + b\ t \\ z = z_\mu + c\ t \\ ax + by + cz + d = 0 \end{cases} \tag{3.1}$$

So, the projection of μ on the plane is given by $\mu^* = \mu + t^* < [a, b, c]^T >$, where t^* (3.2) is the value of the parameter that solves the system (3.1). Finally, the symmetric point μ^s can be obtained using the value $2t^*$ as parameter in the expression of the line (3.3).

$$t^* = -\frac{a\ x_\mu + b\ y_\mu + c\ z_\mu + d}{a^2 + b^2 + c^2} \tag{3.2}$$

$$\mu^s = \mu + 2t^* < a, b, c > \tag{3.3}$$

μ^s will be the mean value of the symmetric Gaussian probability distribution, with respect to the plane $\pi : ax + by + cz + d = 0$. It is worth mentioning that it is possible to define the symmetrization of a point P as an affine transformation: $P^s = H \cdot P + L$. In fact, defining the unit vector orthogonal to the plane as $\left(\boldsymbol{n} := [a\ b\ c]^T / \sqrt{a^2 + b^2 + c^2}\right)$, it follows

$$P^s = P - 2(\boldsymbol{n}^T P + d)\boldsymbol{n} = P - 2\boldsymbol{n}\boldsymbol{n}^T P - 2\boldsymbol{n}d = \left[I - 2\boldsymbol{n}\boldsymbol{n}^T\right] P - 2\boldsymbol{n}d = H \cdot P + L \tag{3.4}$$

where $H := \left[I - 2\boldsymbol{n}\boldsymbol{n}^T\right]$ is a linear transformation that describes a reflection about a plane containing the origin, and it is known as *Householder transformation* [14]. $L := -2\boldsymbol{n}d$ is the translation to be considered when the plane does not contain the origin ($d \neq 0$). Thus, symmetrization with respect to plane $\pi : ax + by + cz + d = 0$ (with l_2-norm of *a, b, c* equal to unity) is given by

$$\begin{bmatrix} x^s \\ y^s \\ z^s \\ 1 \end{bmatrix} = \begin{bmatrix} H & L \\ \mathbf{0}_{1\times 3} & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} 1 - 2a^2 & -2ab & -2ac & -2ad \\ -2ab & 1 - 2b^2 & -2bc & -2bd \\ -2ac & -2bc & 1 - 2c^2 & -2cd \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \tag{3.5}$$

The next passage is to find a way to calculate Σ^s, the symmetric covariance matrix. Matrix Σ is completely defined by its eigenvalues λ_i and eigenvectors $\boldsymbol{v}_i$ ($i = 1, 2, 3$), and the symmetrization of Σ with

respect to a plane can be seen as a rotation of its eigenvectors, keeping the eigenvalues unchanged. The symmetrization approach proposed is based on finding the symmetric eigenvectors $\boldsymbol{v}_i^s$, in order to use them, together with the associated eigenvalues λ_i, to build the new matrix Σ^s, employing the *singular value decomposition*.

It is useful to consider a vector as two points whose difference defines its direction. In this way, the symmetric vector can be obtained from the difference between the symmetric of these two points. In the case in exam, each eigenvector $\boldsymbol{v}_i$ can be defined by the point μ, common to the three, and another obtained summing the vector to μ.

$$P_{\boldsymbol{v}_i} = \mu + \boldsymbol{v}_i \quad \text{for} \quad i = 1, 2, 3 \tag{3.6}$$

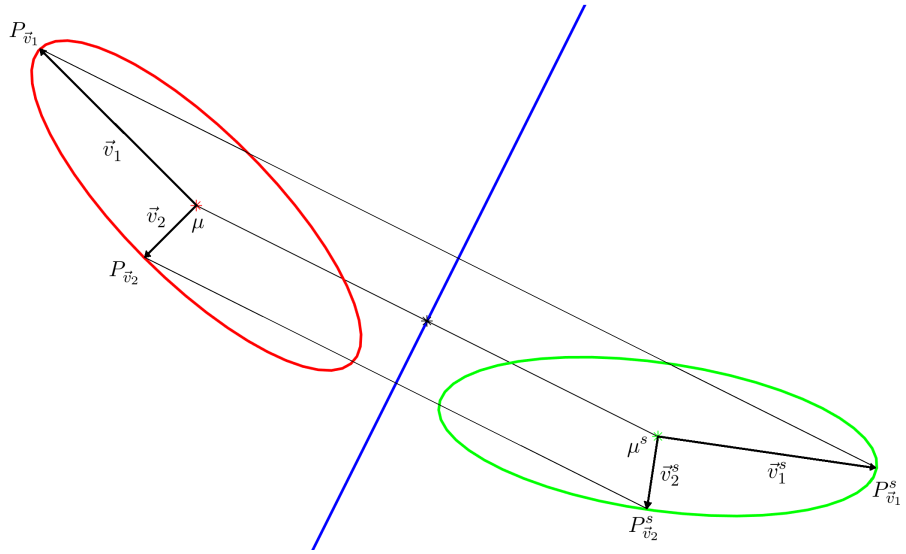


Figure 3.2: *Planar symmetrization method applied to a 2D Gaussian distribution. Note how the symmetric eigenvectors can be easily obtained from the symmetrization of points* μ, $P_{\boldsymbol{v}_1}$ *and* $P_{\boldsymbol{v}_i}$.

The symmetrization procedure described before for μ can be applied to the three points $P_{\boldsymbol{v}_i}$, obtaining $P^s_{\boldsymbol{v}_i}$ for $i = 1, 2, 3$. The relation between points and eigenvectors (3.6) is of course true also in the symmetrized case, resulting in the final expression of the symmetric eigenvectors $\boldsymbol{v}_i^s$:

$$\boldsymbol{v}_i^s = P^s_{\boldsymbol{v}_i} - \mu^s \quad \text{for} \quad i = 1, 2, 3 \tag{3.7}$$

Finally it is possible to obtain the covariance matrix Σ^s associated to the symmetric Gaussian probability density function $\mathcal{N}(\mu^s, \Sigma^s)$ applying the *singular value decomposition* for symmetric matrices, as shown in (3.8).

$$\Sigma^s = V^s \; D \; (V^s)^T \quad \text{where} \quad V^s = [\boldsymbol{v}_1^s | \boldsymbol{v}_2^s | \boldsymbol{v}_3^s] \quad \text{and} \quad D = \begin{bmatrix} \lambda_1 & 0 & 0 \\ 0 & \lambda_2 & 0 \\ 0 & 0 & \lambda_3 \end{bmatrix} \tag{3.8}$$

Algorithm 1: 3D Gaussian Planar Symmetrization

Data: μ, Σ, plane: a, b, c, d

Result: μ^s, Σ^s

find the symmetric of μ *w.r. to plane*:

$t_\mu = -(ax_\mu + by_\mu + cz_\mu + d)/(a^2 + b^2 + c^2)$

$\mu^s = \mu + 2t_\mu[a, b, c]^T$

compute Σ*'s eigenvalues:* λ_1, λ_2, λ_3

compute Σ*'s eigenvectors:* $\boldsymbol{v}_1$, $\boldsymbol{v}_2$, $\boldsymbol{v}_3$

for *i=1,2,3* **do**

  compute the symmetric of eigenvector $\boldsymbol{v}_i$:

  $P_{\boldsymbol{v}_i} = [x_i, y_i, z_i]^T = \mu + \boldsymbol{v}_i$

  $t_{\boldsymbol{v}_i} = -(ax_i + by_i + cz_i + d)/(a^2 + b^2 + c^2)$

  $P^s_{\boldsymbol{v}_i} = \mu^s + 2t_{\boldsymbol{v}_i}[a, b, c]^T$

  $\boldsymbol{v}^s_i =< P^s_{\boldsymbol{v}_i} - \mu^s >$

$V^s := [\boldsymbol{v}^s_1|\boldsymbol{v}^s_2|\boldsymbol{v}^s_3]$

$D := \text{diag}(\lambda_1, \lambda_2, \lambda_3)$

compute the symmetric covariance matrix: $\Sigma^s = V^s D(V^s)^T$

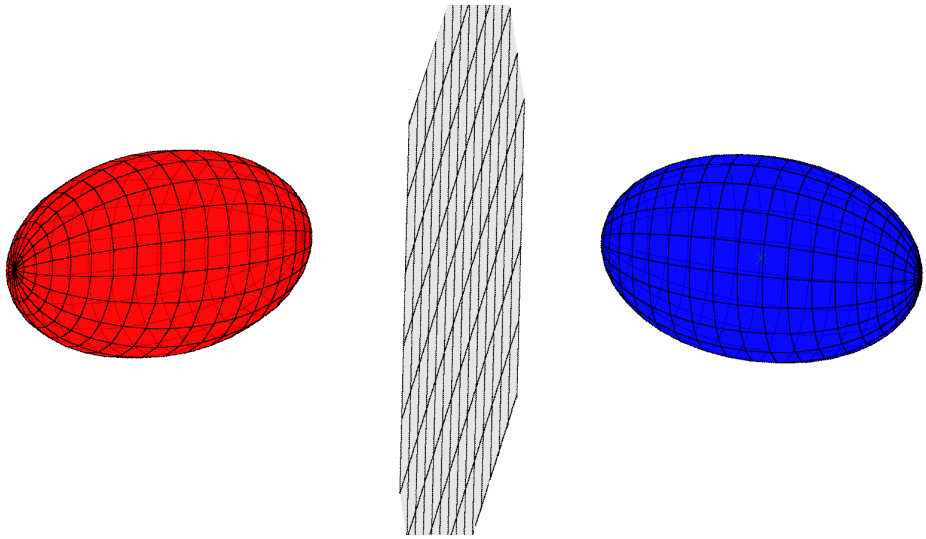


Figure 3.3: *Planar symmetry for 3D Gaussian probability distributions, represented using ellipsoids.*

This immediate "*symmetrization-by-points*" method is possible because Σ^s for planar symmetries have the same eigenvalues as Σ, and only rotated eigenvectors. In fact, in presence of non-planar surfaces some distortion factor has to be taken into account, as it is presented in the next section.

3.1.2 Non-planar symmetries

Considering non-planar symmetries it is necessary to take into account that the symmetrization will cause a deformation to the covariance matrix on the other side of the surface, accordingly to its degree of curvature. In this section a symmetrization procedure that takes into account this characteristic is developed.

The first step is to perform a planar symmetrization, in the same way as described before, considering the tangent plane going through the projection of $\mu = [x_\mu, y_\mu, z_\mu]^T$ on the surface. Let's call the resulting covariance matrix Σ_0^s, this will be the base on which apply a reshaping effect on its eigenvectors and eigenvalues due to the curvature of symmetry surface. The procedure that needs to be followed in order to obtain the tangent plane is explained for the two cases in exam: spherical and cylindrical symmetry.

For a **sphere** of center $C := (x_C, y_C, z_C)$ and radius *R*, the projection of μ onto the surface is the interception between the line going through center *C* and μ, with the sphere, whose direction is given by $\boldsymbol{j} =< [j_x, j_y, j_z]^T >:=< \mu - C >$. This line can be expressed in parametric form as $C + t\ \boldsymbol{j}$. The system used to solve the problem is the following,

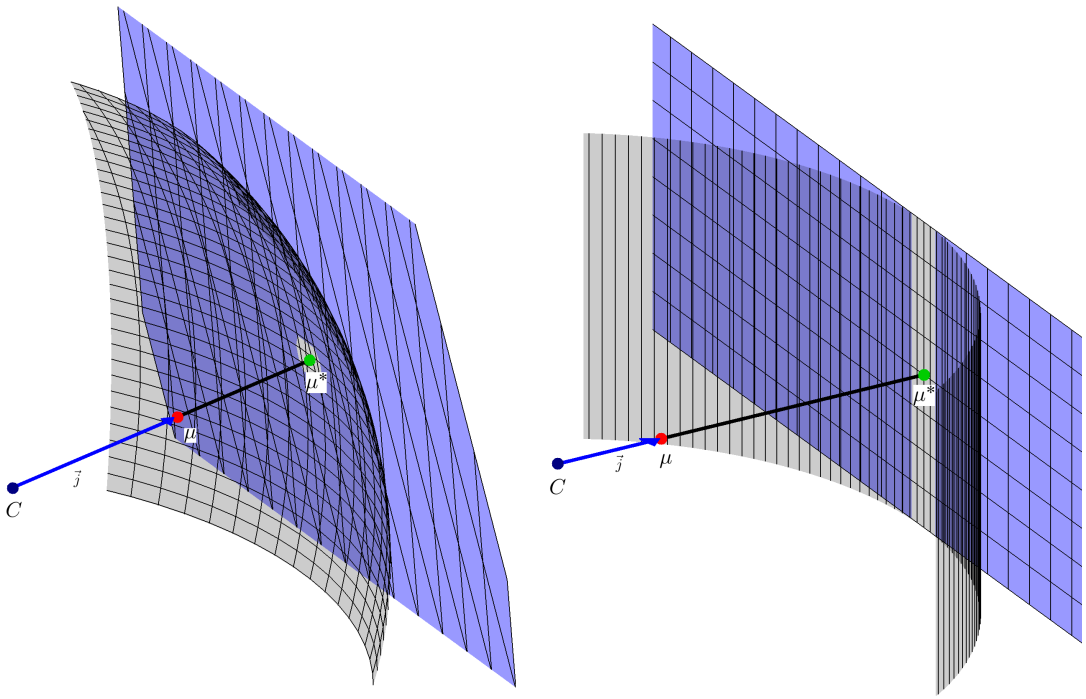
$$\begin{cases} x = x_C + t\ j_x \\ y = y_C + t\ j_y \\ z = z_C + t\ j_z \\ (x - x_C)^2 + (y - y_C)^2 + (z - z_C)^2 = R^2 \end{cases} \tag{3.9}$$

whose solution is given by $x_\mu^* = x_C + t^* j_x$, $y_\mu^* = y_C + t^* j_y$, and $z_\mu^* = z_C + t^* j_z$, with $t^* = R/(j_x^2 + j_y^2 + j_z^2)$. The tangent plane $ax + by + cz + d = 0$ has to be orthogonal to the vector $\boldsymbol{j}$ and the projected point $\mu^* = [x_\mu^*, y_\mu^*, z_\mu^*]^T$ must belong to it, thus, its parameters will be $a = j_x$, $b = j_y$, $c = j_z$, and $d = -ax_\mu^* - by_\mu^* - c_\mu^*$.

On the other hand, for a vertical **cylinder** of center (x_C, y_C) and radius *R*, the plane is tangent to the surface along a vertical line, being the cylinder infinite along that direction. The projection $\mu^* = [x_\mu^*, y_\mu^*, z_\mu^*]^T$ is calculated on the horizontal section cutting the cylinder at μ's height, as the interception of the line going through μ and point $C := (x_C, y_C, z_\mu)$, and the surface. Note how both points defining the line has the same z coordinate, so its direction is given by a vector $\boldsymbol{j} =< [j_x, j_y, j_z]^T >:=< \mu - C >$ whose third component is $j_z = 0$. The system considered to solve the problem is the following,

$$\begin{cases} x = x_C + t\ j_x \\ y = y_C + t\ j_y \\ z = z_C \\ (x - x_C)^2 + (y - y_C)^2 = R^2 \end{cases} \tag{3.10}$$

whose solution is given by $x_\mu^* = x_C + t^* j_x$, $y_\mu^* = y_C + t^* j_y$, and $z_\mu^* = z_C$, with $t^* = R/(j_x^2 + j_y^2)$. Then, similarly to the sphere's case, the tangent plane $ax + by + cz + d = 0$ will be defined by $a = j_x$, $b = j_y$, $c = 0$, and $d = -ax_\mu^* - by_\mu^*$.



(a) *Tangent plane to a sphere* (b) *Tangent plane to a cylinder*

Figure 3.4: *Tangent planes to spherical/cylindrical surface passing through projection* μ^*.

The tangent plane obtained is used to compute a preliminary planar symmetrization, whose results are the mean μ^s, and Σ_0^s. The covariance matrix will be modified taking into account the deformation caused by the curvature in the symmetry surface. If μ is inside the surface Σ^s is amplified with respect to Σ_0^s, if μ is outside, Σ^s is reduced with respect to Σ_0^s.

Eigenvectors $\boldsymbol{v}_i^0$ and eigenvalues λ_i^0 $(i = 1, 2, 3)$ are obtained from Σ_0^s, and it is necessary to modify them to reshape the covariance matrix accordingly to the curve symmetry. The eigenvectors are projected onto the directions defining the curvature: an horizontal vector tangent at the surface for the vertical cylinder, and two orthogonal vectors tangent at the surface in the case of the sphere. Only those components are modified by a scaling factor that depends on the radius *r* and the distance from the surface *d*: $k = \frac{r+d}{r-d}$. Depending on the relative position of μ with respect to the surface, the considered components are multiplied by *k*, if μ is inside the surface, or divided by *k*, if outside. The factor depends on the value of *d* with respect to *r*, its expression is based on the proportionality among segments defined in *Thales' theorem*, as described in Fig. 3.5. Note how in the case of a plane $k = 1$, considering the center of curvature at infinite.

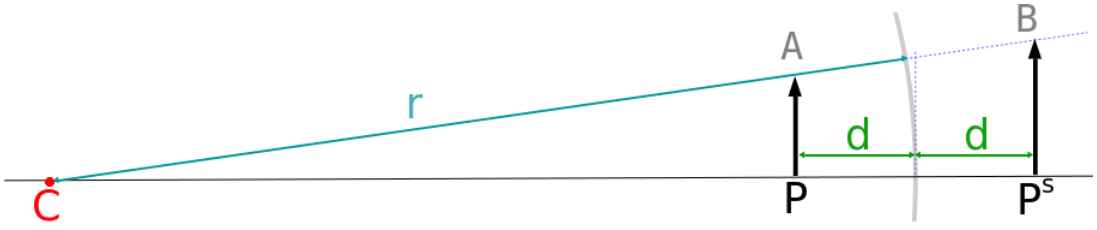


Figure 3.5: *Amplification in magnitude due to curve symmetry surface. The proportionality relationship used to derive k is* $PA : CP = P^sB : CP^s$*, i.e.,* $PA : (r - d) = P^sB : (r + d)$.

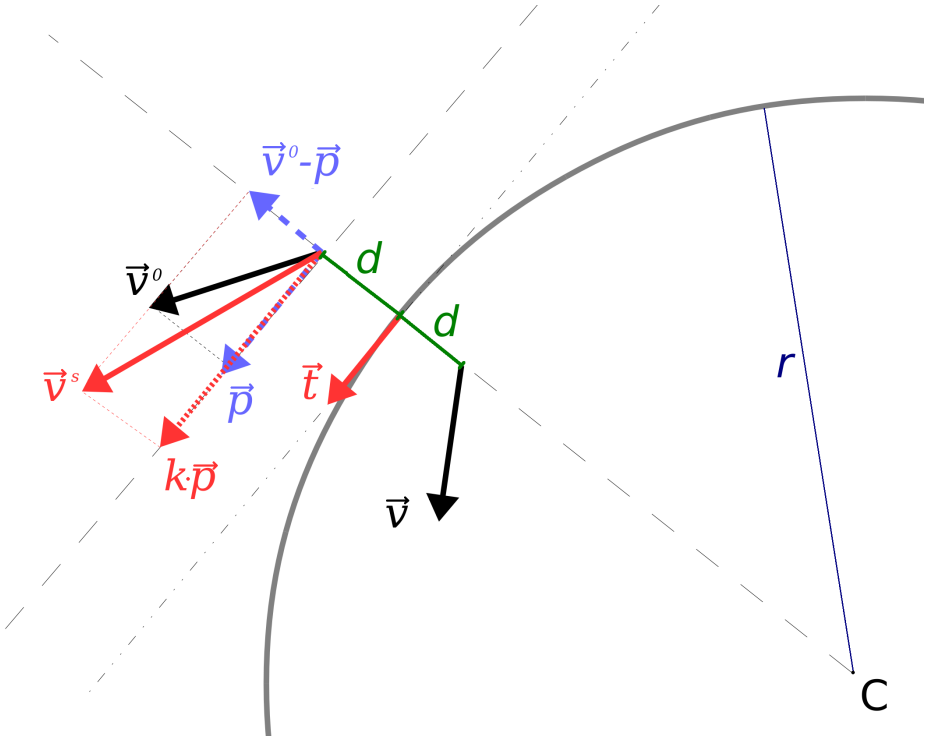


Figure 3.6: *Scheme for the eigenvectors reshaping method in a simplified 2D case.*

The reshaping effect introduced is described with the help of Fig. 3.6, the vector $\boldsymbol{v}_i^0$ (obtained from the initial planar symmetrization, for $i = 1, 2, 3$) is decomposed into the projection $\boldsymbol{p}_i = \frac{\boldsymbol{v}_i^0 \cdot \boldsymbol{t}}{\boldsymbol{t} \cdot \boldsymbol{t}} \cdot \boldsymbol{t}$ onto the curvature direction $\boldsymbol{t}$, and the difference $\boldsymbol{v}_i^0 - \boldsymbol{p}_i$. Then, the component $\boldsymbol{p}_i$ is multiplied by the scaling factor k to obtain the reshaped symmetric eigenvector $\boldsymbol{v}_i^s$, as $\boldsymbol{v}_i^s = k \cdot \boldsymbol{p}_i + (\boldsymbol{v}_i^0 - \boldsymbol{p}_i)$. In the cylindrical case there is only one tangent direction on which apply the scaling factor, the horizontal one defined by $\boldsymbol{t} = [-j_y, j_x, 0]$, instead, with a spherical symmetry surface, the tangent directions to take into account are any two vectors $\boldsymbol{t}_1, \boldsymbol{t}_2$ orthogonal to $\boldsymbol{j}$, taken from its *null space*. In both occasions $\boldsymbol{j} =< [j_x, j_y, j_z]^T >:=< \mu - C >$, as described before.

Then, eigenvalues are also updated multiplying them for the ratio between the norm of the associated reshaped eigenvector and the original one, $\lambda_i^s = \lambda_i^0 \cdot \frac{|\boldsymbol{v}_i^s|}{|\boldsymbol{v}_i|}$. The symmetric covariance matrix is obtained from $\Sigma^s = V^s \; D \; (V^s)^{-1}$, being V^s the matrix whose columns are the new eigenvectors $\boldsymbol{v}_i^s$ normalized and D a matrix with the new eigenvalues λ_i^s on the diagonal and zeros on all the other entries.

This procedure does not guarantee the symmetry of the matrix obtained, because reshaped eigenvectors are no longer orthonormal, hence to ensure symmetry, Σ^s is corrected substituting it with $(\Sigma^s + (\Sigma^s)^T)/2$, averaging each pair of symmetric non-diagonal entries. The effects of the reshaping effect caused by non-planar symmetries can be visualized in Fig. (3.7), where a simple application on a 2D Gaussian distribution is reported.

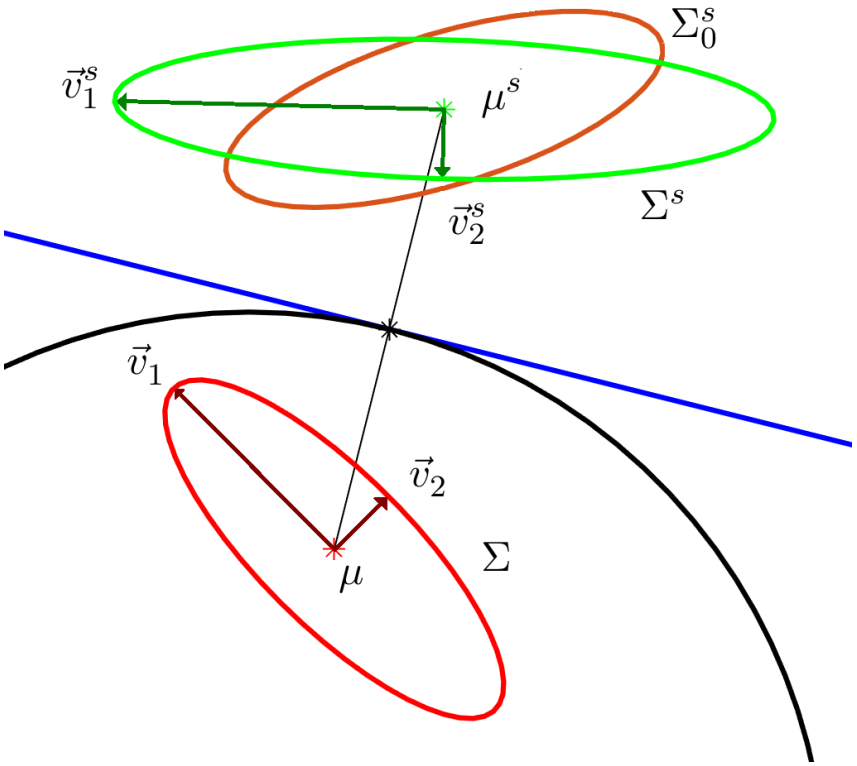


Figure 3.7: *Curve symmetrization method applied to a 2D Gaussian distribution. Note the effects of the reshaping on the eigenvectors of* Σ^s*, with respect to the planar symmetric covariance* Σ_0^s.

Algorithm 2: 3D Gaussian Spheric Symmetrization

Data: μ, Σ, sphere: x_C, y_C, z_C, R

Result: μ^s, Σ^s

find the plane tangent to the sphere passing through projection μ^*:

$a = \mu_x - x_C$, $b = \mu_y - y_C$, $c = \mu_z - z_C$

$t^* = R/(a^2 + b^2 + c^2)$

$\mu^* = C + t^*[a, b, c]^T$

$d = -ax^*_\mu - by^*_\mu - cz^*_\mu$

obtain the initial planar symmetrization:

(μ^s, Σ^s_0) = 3D Gaussian Planar Symmetrization$(\mu, \Sigma, a, b, c, d)$

compute Σ^s_0*'s eigenvalues:* λ^0_1, λ^0_2, λ^0_3

compute Σ^s_0*'s eigenvectors:* $\boldsymbol{v}^0_1$, $\boldsymbol{v}^0_2$, $\boldsymbol{v}^0_3$

compute distance: $D = ||\mu^* - \mu||$

scaling factor:

if (D<R) $k = (R + D)/(R - D)$

else $k = (R - D)/(R + D)$

obtain orthogonal vectors $\boldsymbol{t}_1, \boldsymbol{t}_2$ *from* **null**$(< [a, b, c]^T >)$

for *i=1,2,3* **do**

- *compute distorted eigenvector and eigenvalue:*
- $\boldsymbol{p}_1 = (\boldsymbol{v}^0_i \cdot \boldsymbol{t}_1)/(\boldsymbol{t}_1 \cdot \boldsymbol{t}_1) \cdot \boldsymbol{t}_1$
- $\boldsymbol{p}_2 = (\boldsymbol{v}^0_i \cdot \boldsymbol{t}_2)/(\boldsymbol{t}_2 \cdot \boldsymbol{t}_2) \cdot \boldsymbol{t}_2$
- $\boldsymbol{v}^s_i = k\boldsymbol{p}_1 + k\boldsymbol{p}_2 + (\boldsymbol{v}^0_i - \boldsymbol{p}_1 - \boldsymbol{p}_2)$
- $\lambda^s_i = \lambda^0_i \cdot \frac{|\boldsymbol{v}^s_i|}{|\boldsymbol{v}_i|}$
- *normalize* $\boldsymbol{v}^s_i$

$V^s := [\boldsymbol{v}^s_1 | \boldsymbol{v}^s_2 | \boldsymbol{v}^s_3]$

$D := \text{diag}(\lambda^s_1, \lambda^s_2, \lambda^s_3)$

compute the symmetric covariance matrix: $\Sigma^s = V^s D (V^s)^{-1}$

force symmetry: $\Sigma^s = (\Sigma^s + (\Sigma^s)^T)/2$

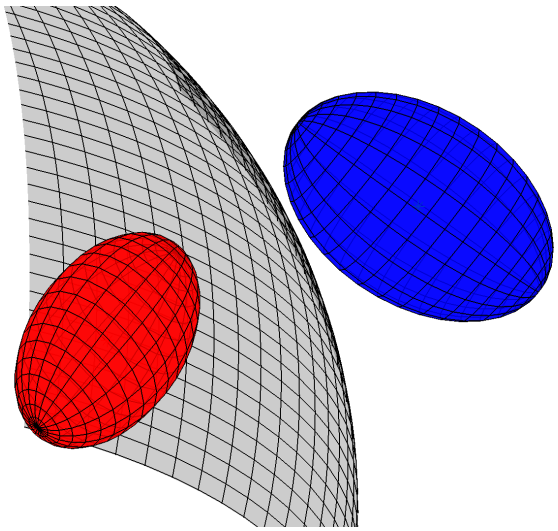


Figure 3.8: *Spherical symmetry for 3D Gaussian probability distributions, represented using ellipsoids.*

Algorithm 3: 3D Gaussian Cylindrical Symmetrization

Data: μ, Σ, cylinder: x_C, y_C, R

Result: μ^s, Σ^s

find the plane tangent to the cylinder passing through projection μ^*:

$a = \mu_x - x_C$, $b = \mu_y - y_C$, $c = 0$

$t^* = R/(a^2 + b^2)$

$\mu^* = C + t^*[a, b, c]^T$

$d = -ax^*_\mu - by^*_\mu$

obtain the initial planar symmetrization:

(μ^s, Σ^s_0) = 3D Gaussian Planar Symmetrization$(\mu, \Sigma, a, b, c, d)$

compute Σ^s_0*'s eigenvalues:* λ^0_1, λ^0_2, λ^0_3

compute Σ^s_0*'s eigenvectors:* $\boldsymbol{v}^0_1$, $\boldsymbol{v}^0_2$, $\boldsymbol{v}^0_3$

compute distance: $D = ||\mu^* - \mu||$

scaling factor:

if (D<R) $k = (R + D)/(R - D)$

else $k = (R - D)/(R + D)$

obtain orthogonal vector $\boldsymbol{t} =< [-b, a, 0]^T >$

for *i=1,2,3* **do**

  compute distorted eigenvector and eigenvalues:

  $\boldsymbol{p} = (\boldsymbol{v}^0_i \cdot \boldsymbol{t})/(\boldsymbol{t} \cdot \boldsymbol{t}) \cdot \boldsymbol{t}$

  $\boldsymbol{v}^s_i = k\boldsymbol{p} + (\boldsymbol{v}^0_i - \boldsymbol{p})$

  $\lambda^s_i = \lambda^0_i \cdot \frac{|\boldsymbol{v}^s_i|}{|\boldsymbol{v}_i|}$

  normalize $\boldsymbol{v}^s_i$

$V^s := [\boldsymbol{v}^s_1|\boldsymbol{v}^s_2|\boldsymbol{v}^s_3]$

$D :=$ diag$(\lambda^s_1, \lambda^s_2, \lambda^s_3)$

compute the symmetric covariance matrix: $\Sigma^s = V^s D (V^s)^{-1}$

force symmetry: $\Sigma^s = (\Sigma^s + (\Sigma^s)^T)/2$

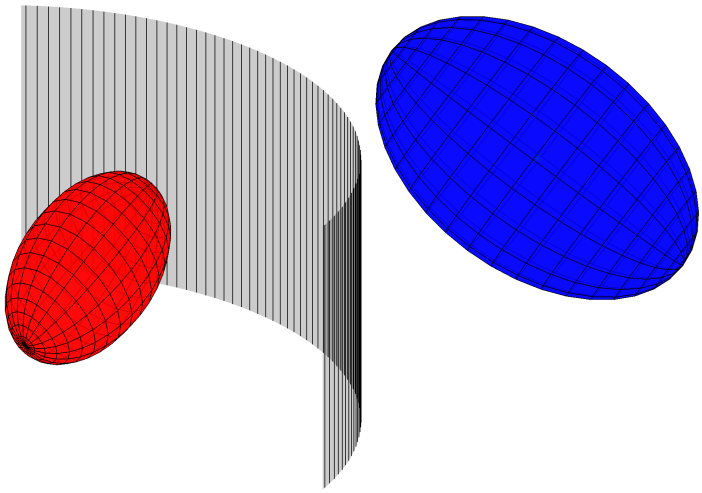


Figure 3.9: *Cylindrical symmetry for 3D Gaussian probability distributions, represented using ellipsoids.*

3.2 Probabilistic movement primitives symmetrization

It is now possible to apply the methods proposed in the previous section to the probabilistic representation of a robot trajectory, in order to extract a symmetric ProMP for the second arm, from the ProMP of the first. The same notation adopted in chapter 2 has been used to describe the ProMPs taken into account.

Recording some demonstrations of the symmetric bimanual task two ProMPs can be built: ProMP$_1$ describing trajectory of the first arm $\boldsymbol{\tau}^1 = \left\{[x_t^1,\ y_t^1,\ z_t^1]^T\right\}_{t=1\dots N_t}$, and ProMP$_2$ describing trajectory of the second $\boldsymbol{\tau}^2 = \left\{[x_t^2,\ y_t^2,\ z_t^2]^T\right\}_{t=1\dots N_t}$. Each robot movement is always given with respect to its own reference frame.

$$\boldsymbol{\tau}_t^i := \begin{bmatrix} x_t^i \\ y_t^i \\ z_t^i \end{bmatrix} = \begin{bmatrix} \Phi_t & 0 & 0 \\ 0 & \Phi_t & 0 \\ 0 & 0 & \Phi_t \end{bmatrix}^T \begin{bmatrix} \boldsymbol{w}_i^x \\ \boldsymbol{w}_i^y \\ \boldsymbol{w}_i^z \end{bmatrix} := \boldsymbol{\Phi}_t^T \boldsymbol{w}_i \qquad \boldsymbol{w}_i \sim \mathcal{N}(\mu_{\boldsymbol{w}}^i, \Sigma_{\boldsymbol{w}}^i) \quad i = 1, 2 \tag{3.11}$$

At each time instant it is possible to determine a 3D Gaussian probability distribution describing the trajectories modeled by the ProMP$_1$.

$$p(\boldsymbol{\tau}_t^1|\boldsymbol{w}_1) = \mathcal{N}(\boldsymbol{\tau}_t^1|\mu_t^1, \Sigma_t^1) := \mathcal{N}(\boldsymbol{\tau}_t^1|\boldsymbol{\Phi}_t^T\mu_{\boldsymbol{w}}^1, \boldsymbol{\Phi}_t^T\Sigma_{\boldsymbol{w}}^1\boldsymbol{\Phi}_t) \quad \text{for} \quad t = 1 \dots N_t \tag{3.12}$$

Hence, a sequence of symmetric distributions $\mathcal{N}(\boldsymbol{\tau}_t^s|\mu_t^s, \Sigma_t^s)$ can be computed from $\mathcal{N}(\boldsymbol{\tau}_t^1|\mu_t^1, \Sigma_t^s)$ (for $t = 1 \dots N_t$) using the methods exposed in section 3.1 and carrying out a change of frame to express trajectory points in the reference frame of the second robot. At each time instant, $\boldsymbol{\tau}_t^1$ is given as a mean point and associated covariance matrix, and from it a symmetric mean point and covariance are computed (depending on the symmetry surface considered). The objective is to use this new sequence of Gaussians to build ProMP$_1^S$ ($\boldsymbol{w}_s \sim \mathcal{N}(\mu_{\boldsymbol{w}}^s, \Sigma_{\boldsymbol{w}}^s)$), symmetrization of the first arm's ProMP.

Considering the expected value of the symmetric trajectory $\mu_t^s = \mathbb{E}[x_t^s, y_t^s, z_t^s]^T$, define the vector of positions for each direction as $\mathbf{x}^s := \mathbb{E}[x_t^s \cdots x_{N_t}^s]$, $\mathbf{y}^s := \mathbb{E}[y_t^s \cdots y_{N_t}^s]$, and $\mathbf{z}^s := \mathbb{E}[z_t^s \cdots z_{N_t}^s]$, and the block basis function matrix $\Psi := [\Phi_1 \dots \Phi_{N_t}]$. $\mu_{\boldsymbol{w}}^s$ can be derived directly like that:

$$\begin{bmatrix} \mathbf{x}^s \\ \mathbf{y}^s \\ \mathbf{z}^s \end{bmatrix} = \boldsymbol{\Psi}^T\mu_{\boldsymbol{w}}^s \quad \text{with} \quad \boldsymbol{\Psi}^T := \begin{bmatrix} \Psi & 0 & 0 \\ 0 & \Psi & 0 \\ 0 & 0 & \Psi \end{bmatrix}^T \tag{3.13}$$

$$\mu_{\boldsymbol{w}}^s = (\boldsymbol{\Psi}^T)^+ \begin{bmatrix} \mathbf{x}^s \\ \mathbf{y}^s \\ \mathbf{z}^s \end{bmatrix} \tag{3.14}$$

For what concerns the computation of $\Sigma_{\boldsymbol{w}}^s$, N_t different equations $\Sigma_t^s = \boldsymbol{\Phi}_t^T\Sigma_{\boldsymbol{w}}^s\boldsymbol{\Phi}_t$ must be taken into account, one for each instant of time. Two different approaches have been considered to compute a unique value of $\Sigma_{\boldsymbol{w}}^s$ from the N_t equations. The first obtains the weight covariance matrix solving $\Sigma_t^s = \boldsymbol{\Phi}_t^T\Sigma_{\boldsymbol{w}}^s\boldsymbol{\Phi}_t$ at each time step and then averages all of them. The second embeds those equations for all time step in a unique block matrix equation, to solve it using *Moore-Penrose pseudoinverse*.

- ***1***: $\Sigma_{\boldsymbol{w}}^s(t) = (\boldsymbol{\Phi}_t^T)^+\Sigma_{\boldsymbol{w}}^s(\boldsymbol{\Phi}_t)^+$ for $t = 1 \dots N_t$ and $\Sigma_{\boldsymbol{w}}^s = \frac{1}{N_t}\sum_t \Sigma_{\boldsymbol{w}}^s(t)$

- ***2:*** $\begin{bmatrix} \vdots \\ \Sigma^s_{\boldsymbol{w}}(t)(\boldsymbol{\Phi}_t)^+ \\ \vdots \end{bmatrix} = \begin{bmatrix} \vdots \\ \boldsymbol{\Phi}_t^T \\ \vdots \end{bmatrix} \Sigma^s_{\boldsymbol{w}}, \quad \text{and} \quad \Sigma^s_{\boldsymbol{w}} = \begin{bmatrix} \vdots \\ \boldsymbol{\Phi}_t^T \\ \vdots \end{bmatrix}^+ \begin{bmatrix} \vdots \\ \Sigma^s_{\boldsymbol{w}}(t)(\boldsymbol{\Phi}_t)^+ \\ \vdots \end{bmatrix}$ for $t = 1 \dots N_t$

The second method requires to modify the matrix $\Sigma^s_{\boldsymbol{w}}$ at the end, to guarantee its symmetry, otherwise it cannot be a valid covariance matrix. This is done by substituting it with $(\Sigma^s_{\boldsymbol{w}} + (\Sigma^s_{\boldsymbol{w}})^T)/2$, averaging each pair of symmetric non-diagonal entries.

In order to choose the most appropriate method to calculate the weight covariance matrix of ProMP$_1^S$, the resulting p.d.f. $\mathcal{N}(\mu^s_{\boldsymbol{w}}, \Sigma^s_{\boldsymbol{w}})$ and the ProMP$_2$ distribution $\mathcal{N}(\mu^2_{\boldsymbol{w}}, \Sigma^2_{\boldsymbol{w}})$ are compared applying both methods to various cases of demonstrated trajectories. To do so, the **Kullback-Leibler divergence** (KL divergence) is used, which is an indicator of the difference between two probability distributions over a random variable [15] .

For multivariate Gaussian distribution (as weights of ProMPs are described) KL divergence can be computed using the formula (3.15).

$$KL(\mathcal{N}_0||\mathcal{N}_1) = \frac{1}{2}\left(\text{tr}(\Sigma_1^{-1}\Sigma_0) + (\mu_1 - \mu_0)^T\Sigma_1^{-1}(\mu_1 - \mu_0) - D + \ln\frac{det(\Sigma_1)}{det(\Sigma_0)}\right) \tag{3.15}$$

considering $\mathcal{N}_0$ as the Gaussian distribution associated to the weights of ProMP$_2$ and $\mathcal{N}_1$ as the probability distribution of the weights of ProMP$_1^s$. D is the dimension of the Gaussian distribution, i.e., $D = 3 \cdot M$ where M is the number of basis functions considered.

The first method to compute $\Sigma^s_{\boldsymbol{w}}$ obtains a bigger KL divergence, so it has been chosen the second method for ProMP symmetrization. The same conclusion can also be reached by looking at the variances of the mean trajectories derived from the different ProMPs (Fig. 3.10) and noting that the first method underestimate the variance, while the second gives a result closer to the variance of ProMP$_2$.

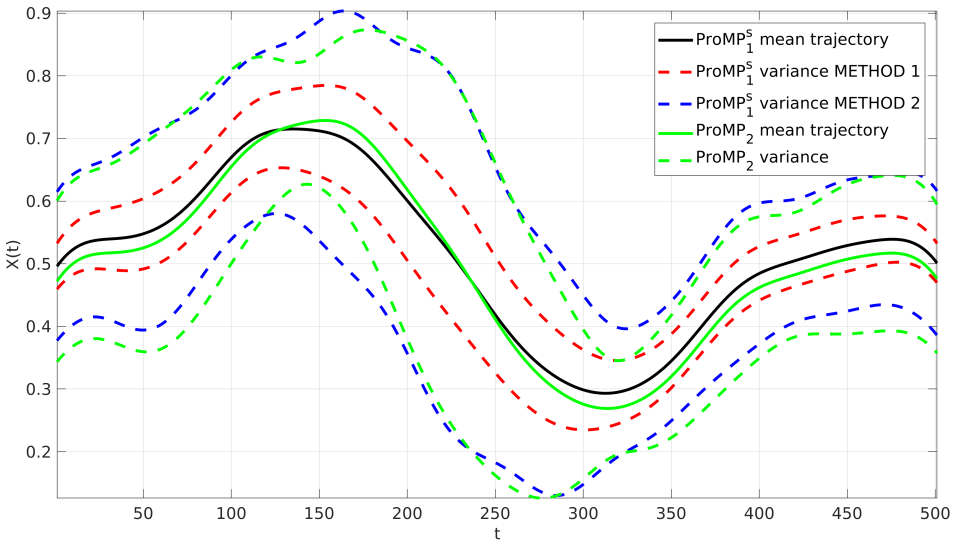


Figure 3.10: *X trajectories and variances for ProMP$_2$ and the ProMP$_1^s$, with the variances computed with both methods.*

The ProMP symmetrization method developed can be applied in all the kind of symmetries considered, planar and curve. It depends only on the time-dependent probabilistic representation of the symmetric trajectory, regardless of whether it derives from a planar, spherical or cylindrical Gaussian symmetrization. Example of some symmetric ProMPs rollouts are showed in Fig. 3.11 - 3.12 - 3.13.

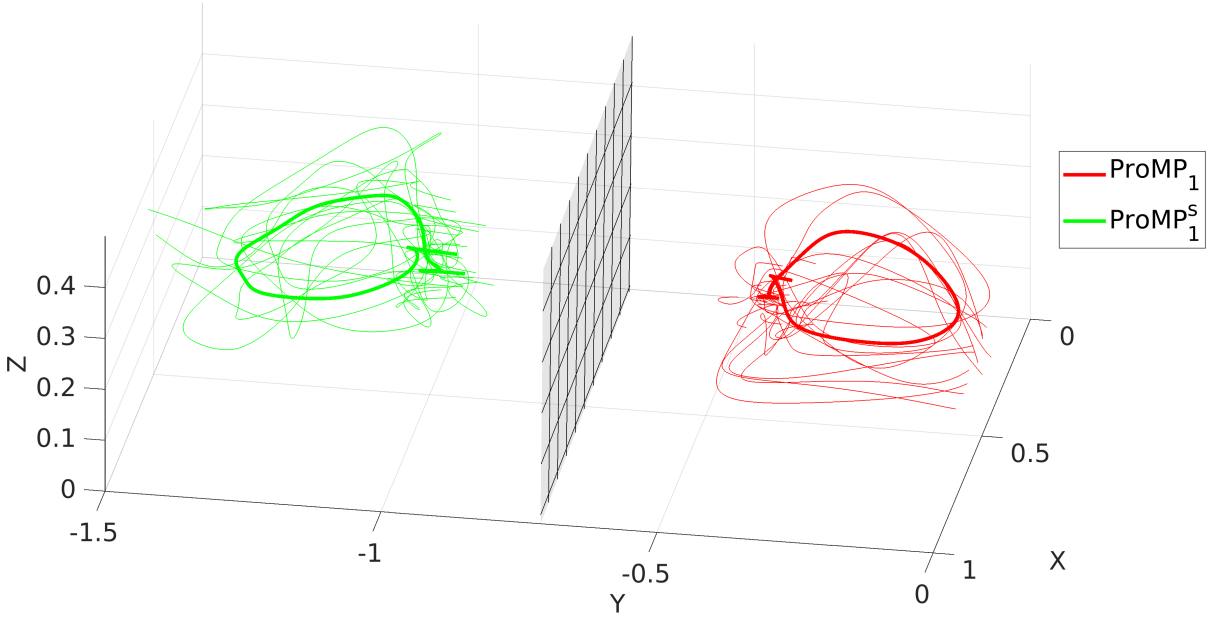


Figure 3.11: *Mean trajectories and rollouts for ProMP$_1$ and ProMP$_1^S$ in presence of a planar symmetry.*

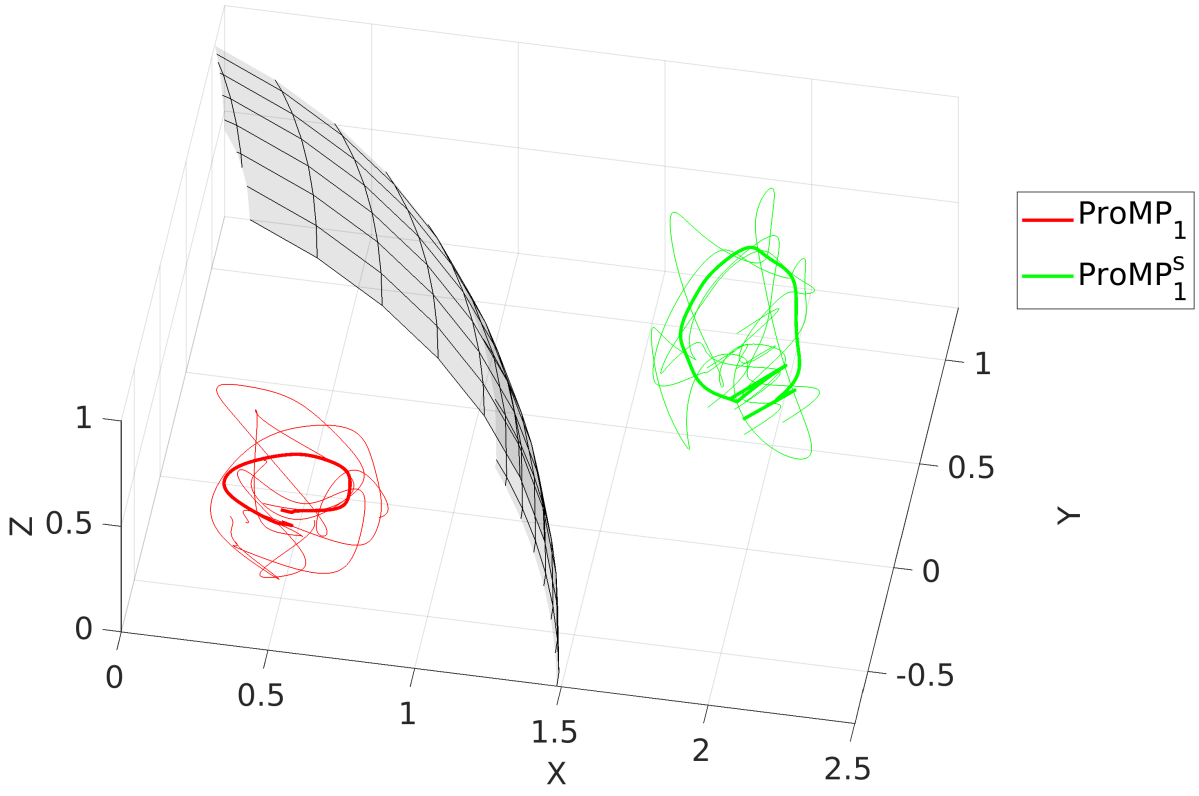


Figure 3.12: *Mean trajectories and rollouts for ProMP$_1$ and ProMP$_1^S$ in presence of a spherical symmetry.*

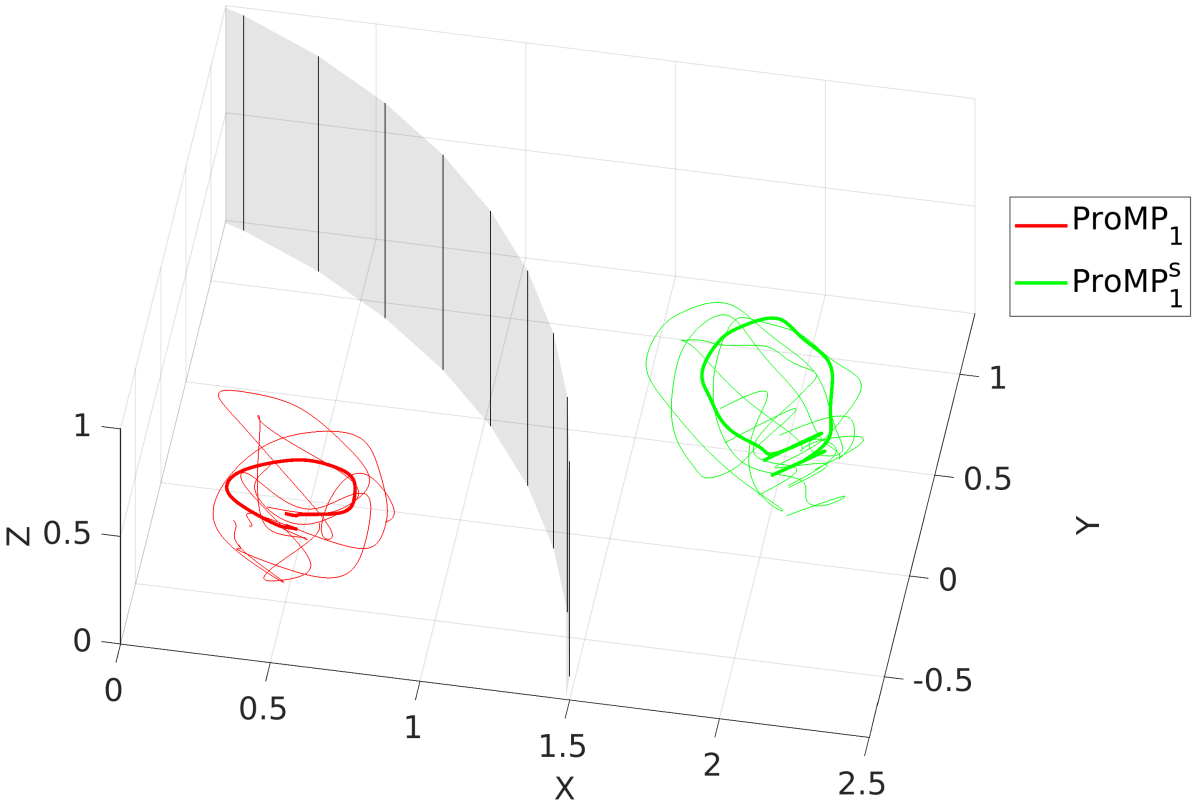


Figure 3.13: *Mean trajectories and rollouts for ProMP$_1$ and ProMP$_1^S$ in presence of a cylindrical symmetry.*

A structured method to obtain symmetric ProMPs for a given symmetry surface, whether it is a plane, a sphere or a cylinder, has been developed. Now it is necessary to obtain a way to estimate a possible symmetry surface from the demonstrated movements, otherwise all those methods cannot be used to facilitate the learning process. This problem will be faced in the next chapter.

4

Estimation of Symmetry Surfaces

In the case of a perfect symmetry, trajectory midpoints, i.e., the mean between the positions of the end-effectors at each time step, would all belong to the same surface. Actually, with data measured from real demonstrations, the symmetry cannot be absolutely perfect, but the distributions of midpoints in the space can be analyzed in order to get the surface that best fit the data. In this chapter estimation approaches for each type of symmetry surface considered in this work (plane, sphere, cylinder) are developed. It is assumed a normal distribution for the parameters and their estimation is given with a mean and a covariance, that can be used for succeeding policy exploration and learning.

4.1 Symmetry plane

The method proposed to estimate the parameters of symmetry plane of a bimanual task, is based on the study of midpoints between the mean trajectories demonstrated with the two robots at each instant of time $\mu_t^1 = [x_t^1, y_t^1, z_t^1]^T$, $\mu_t^2 = [x_t^2, y_t^2, z_t^2]^T$ for $t = 1 \dots N_t$.

$$\boldsymbol{m}_t = \frac{\mu_t^1 + \mu_t^2}{2} \qquad t = 1 \dots N_t \tag{4.1}$$

Principal Component Analysis (PCA) [16] has been used to study the set of midpoints and find a viable symmetry plane. PCA is defined as an orthogonal linear transformation that transforms the data (represented as a matrix $\mathbf{X}$) to a new coordinate system such that the greatest variance by some projection of the data comes to lie on the first coordinate (called the first principal component), the second greatest variance on the second coordinate, and so on. Principal components transformation can be obtained from the *singular value decomposition* of the data matrix: $\mathbf{X} = \mathbf{USV}^T$, where $\mathbf{S}$ is the diagonal matrix of singular values, $\mathbf{V}$ are principal components. In a perfectly symmetric bimanual movement, all the midpoints must belong to the same plane, and PCA would results in a third components describing no variance of data. Thus, dealing with real demonstrations, a possible symmetry plane can be defined as the plane orthogonal to the vector defining the third principal component (direction with the least variation) $\boldsymbol{v}_3$, passing through the center of midpoints $\mu_m := \frac{1}{N_t}\sum_t^{N_t} \boldsymbol{m}_t$ (Fig. 4.1). So, the estimated parameters of the symmetry plane equation $\hat{a}\ x + \hat{b}\ y + \hat{c}\ z + \hat{d} = 0$ are

$$\hat{a} = x_{\boldsymbol{v}_3},\ \hat{b} = y_{\boldsymbol{v}_3},\ \hat{c} = z_{\boldsymbol{v}_3},\ \hat{d} = -a\ x_{\mu_m} - b\ y_{\mu_m} - c\ z_{\mu_m} \tag{4.2}$$

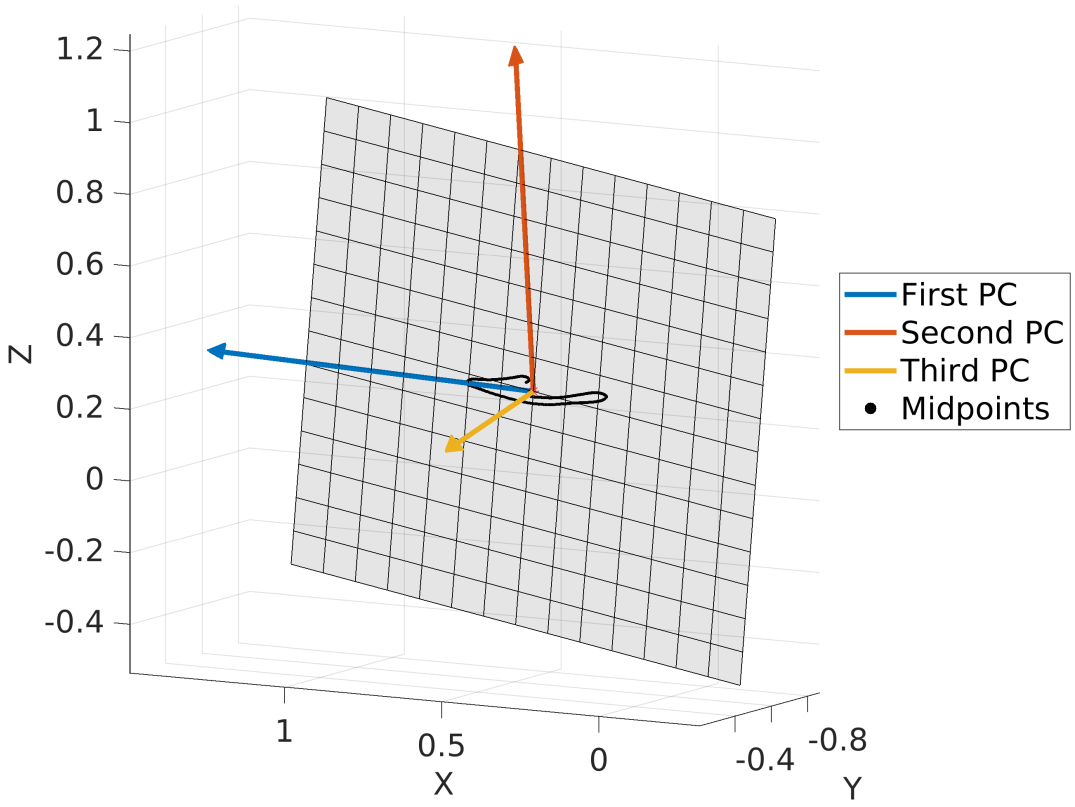


Figure 4.1: *Symmetry plane estimation using PCA.*

It is also possible to calculate also the covariance matrix associated to the estimates of the parameters. This can be done applying PCA method to each pair of demonstrated trajectories independently, obtaining a set of parameters $\{\hat{a}_k, \hat{b}_k, \hat{c}_k, \hat{d}_k\}$ for $k = 1 \dots K_d$. Covariance Σ_π for plane's parameters can be computed with UMLE:

$$\Sigma_\pi = \frac{1}{K_d - 1} \sum_{k=1}^{K_d} \begin{bmatrix} (\hat{a}_k - \hat{a}) \\ (\hat{b}_k - \hat{b}) \\ (\hat{c}_k - \hat{c}) \\ (\hat{d}_k - \hat{d}) \end{bmatrix} \cdot \begin{bmatrix} (\hat{a}_k - \hat{a}) & (\hat{b}_k - \hat{b}) & (\hat{c}_k - \hat{c}) & (\hat{d}_k - \hat{d}) \end{bmatrix} \tag{4.3}$$

4.2 Symmetry sphere

The spheric symmetry is defined by 4 parameters $[x_C, y_C, z_C, R]$, center and radius of the sphere. To perform their estimation, the lines $l_{t,k}$ connecting two end-effectors at each time instant, for every demonstrations, are taken into account. The center of the sphere is chosen as the point with minimum quadratic distance from all the lines $l_{t,k}$:

$$\hat{C} = (\hat{x_C}, \hat{y_C}, \hat{z_C}) = \underset{P:=(x,y,z)}{\operatorname{argmin}} \sum_{\forall t,k} d^2(P, l_{t,k}) \tag{4.4}$$

Then, the radius can be obtained as the distance between $\hat{C}$ and the center of demonstrations' midpoints μ_m: $\hat{R} = d(\hat{C}, \mu_m)$.

Like in the case of the plane, it is possible to calculate also the covariance matrix associated to the estimates of parameters. This can be done finding the center and radius of the sphere for each pair of demonstrated trajectories independently, obtaining in this way a set of parameters $\{\hat{x_C}^k, \hat{y_C}^k, \hat{z_C}^k, \hat{R}^k\}$ for $k = 1 \dots K_d$. Covariance Σ_σ for sphere's parameters can be computed with UMLE:

$$\Sigma_\sigma = \frac{1}{K_d - 1} \sum_{k=1}^{K_d} \begin{bmatrix} (\hat{x_C}^k - \hat{x_C}) \\ (\hat{y_C}^k - \hat{y_C}) \\ (\hat{z_C}^k - \hat{z_C}) \\ (\hat{R}^k - \hat{R}) \end{bmatrix} \cdot \begin{bmatrix} (\hat{x_C}^k - \hat{x_C}) & (\hat{y_C}^k - \hat{y_C}) & (\hat{z_C}^k - \hat{z_C}) & (\hat{R}^k - \hat{R}) \end{bmatrix} \tag{4.5}$$

4.3 Symmetry cylinder

The cylindric symmetry is defined by 3 parameters $[x_c, y_c, r]$, 2D center and radius of the cylinder. Like in the spherical case, the lines $l_{t,k}$ connecting two end-effectors at each time instant, for every demonstrations, are used, but considering only XY coordinates (being Z not changed by cylindrical symmetrization). The center of the cylinder is chosen as the point with minimum quadratic distance from all the lines $l_{t,k}$:

$$\hat{C} = (\hat{x_C}, \hat{y_C}) = \underset{P:=(x,y)}{\operatorname{argmin}} \sum_{\forall t,k} d^2(P, l_{t,k}) \tag{4.6}$$

Radius can be obtained as the distance between $\hat{C}$ and the center of demonstrations' midpoints μ_m, considering only XY coordinates: $\hat{R} = d\left(\hat{C}, \begin{bmatrix} x_{\mu_m} \\ y_{\mu_m} \end{bmatrix}\right)$.

Like in the previous cases, the covariance matrix associated to the estimates of parameters is calculated, from the cylinder's center and radius obtained considering each pair of demonstrated trajectories independently, resulting in a set of parameters $\{\hat{x_C}^k, \hat{y_C}^k, \hat{R}^k\}$ for $k = 1 \dots K_d$. Covariance Σ_γ for cylinder's parameters can be computed with UMLE:

$$\Sigma_\gamma = \frac{1}{K_d - 1} \sum_{k=1}^{K_d} \begin{bmatrix} (\hat{x_C}^k - \hat{x_C}) \\ (\hat{y_C}^k - \hat{y_C}) \\ (\hat{R}^k - \hat{R}) \end{bmatrix} \cdot \begin{bmatrix} (\hat{x_C}^k - \hat{x_C}) & (\hat{y_C}^k - \hat{y_C}) & (\hat{R}^k - \hat{R}) \end{bmatrix} \tag{4.7}$$

4.4 Optimization of surface's parameters

The surface estimated from demonstrated movements using the methods explained, could not be the optimal one to have the better symmetrization possible, hence this initial estimation needs to be refined using optimization methods. To do so, the Kullback–Leibler divergence is used again as a measure of how the ProMP$_2$ weights probability distribution diverges from the one obtained with the symmetrization of ProMP$_1$. KL divergence is computed using multivariate Gaussian formula (3.15), as follows.

$$KL(\mathcal{N}_2 || \mathcal{N}_1^s) = \frac{1}{2} \left(\text{tr}[(\Sigma_{\boldsymbol{w}}^s)^{-1} \Sigma_{\boldsymbol{w}}^2] + (\mu_{\boldsymbol{w}}^s - \mu_{\boldsymbol{w}}^2)^T (\Sigma_{\boldsymbol{w}}^s)^{-1} (\mu_{\boldsymbol{w}}^s - \mu_{\boldsymbol{w}}^2) - 3 \cdot M + \ln \frac{det(\Sigma_{\boldsymbol{w}}^s)}{det(\Sigma_{\boldsymbol{w}}^2)} \right) \tag{4.8}$$

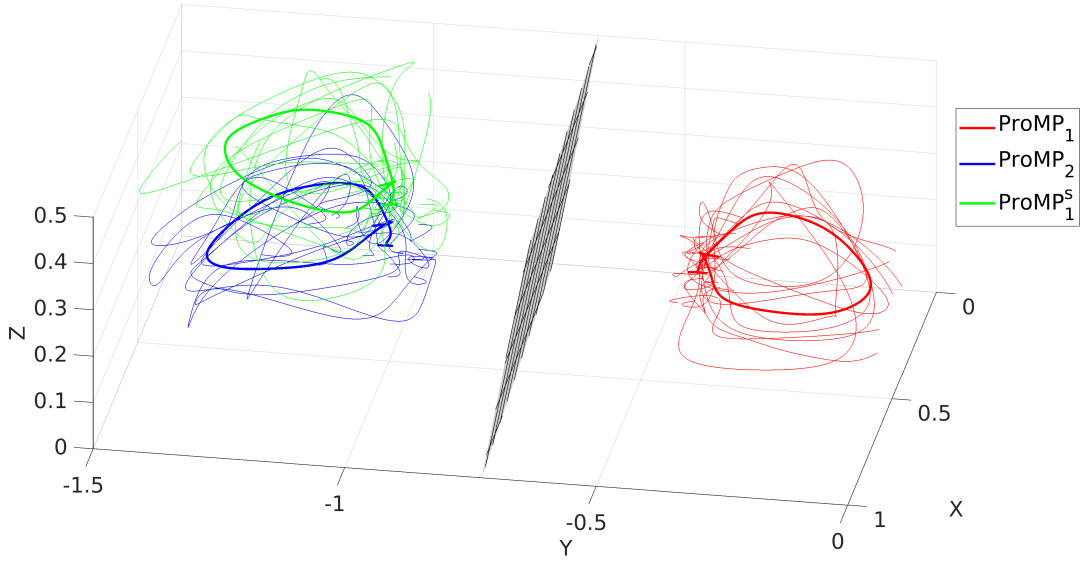
The surface parameters are optimized to minimize the resulting KL divergence between ProMP_1^s and ProMP_2 distributions.

$$\{\hat{a}, \hat{b}, \hat{c}, \hat{d}\}^{\text{updated}} = \underset{a,b,c,d}{\text{argmin}} KL(\mathcal{N}_2 || \mathcal{N}_1^s(a, b, c, d)) \tag{4.9}$$

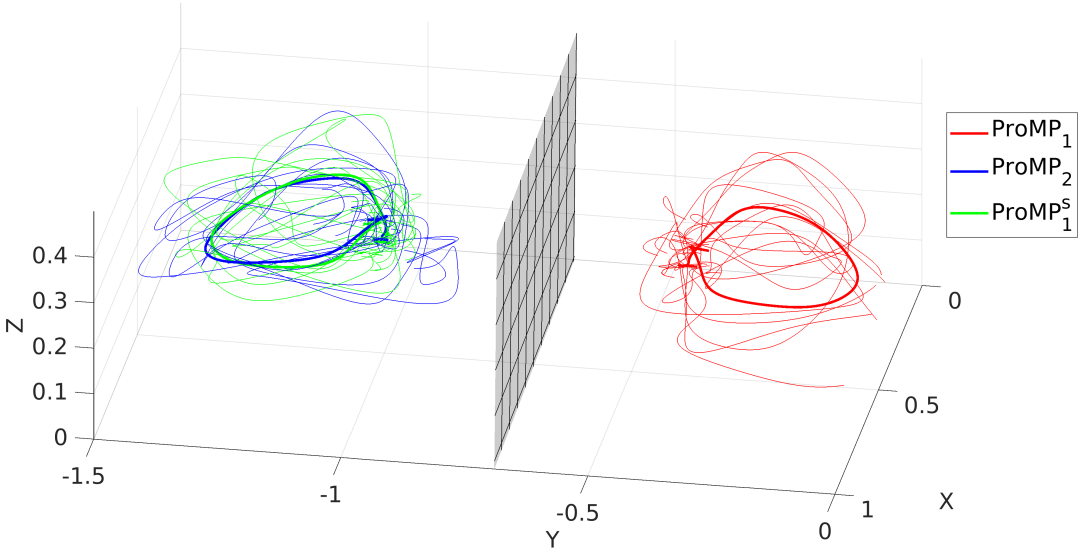
$$\{\hat{x_C}, \hat{y_C}, \hat{z_C}, \hat{R}\}^{\text{updated}} = \underset{x_C,y_C,z_C,R}{\text{argmin}} KL(\mathcal{N}_2 || \mathcal{N}_1^s(x_C, y_C, z_C, R)) \tag{4.10}$$

$$\{\hat{x_C}, \hat{y_C}, \hat{R}\}^{\text{updated}} = \underset{x_C,y_C,R}{\text{argmin}} KL(\mathcal{N}_2 || \mathcal{N}_1^s(x_C, y_C, R)) \tag{4.11}$$

As an example, in Fig. 4.2 it is showed the effect of parameters' optimization on the case of a planar symmetry in the bimanual movement. Note how the ProMP_1^s obtained with the optimal symmetry plane imitates almost perfectly the ProMP_2.



(a) *Initial symmetry estimation*



(b) *Optimized symmetry estimation*

Figure 4.2: *Mean trajectories and rollouts for ProMP_1, ProMP_2, and ProMP_1^s, before and after optimization of the symmetry plane.*

5

Reinforcement Learning Policy Search for Symmetric Bimanual Tasks

The symmetrization methods developed are used to represent bimanual robotic tasks with a reduced amount of parameters defining the policy. In fact, instead of using two distinct ProMPs to model the movement of both arms separately, only a single ProMP can be built and updated for one of the two robots, and then, applying ProMP-symmetrization method, MP for the second arm is obtained. The aim is to test if using this symmetrization method in the learning process can lead to a faster convergence in the policy search.

5.1 Relative Entropy Policy Search

Relative Entropy Policy Search (REPS) [17] is an algorithm that aims to find the policy π^* that maximizes the expected reward for a given task. The REPS algorithm uses in its definition the Kullback–Leibler divergence, that is also called *relative entropy*, hence the name of the algorithm. KL divergence is defined as a non-symmetric indicator of the difference between two probability distributions *p*, *q* over a random variable *x*, obtained from:

$$\mathrm{KL}(p||q) = \int p(x) log \frac{p(x)}{q(x)} dx \tag{5.1}$$

A *ProMP* policy $\pi(\boldsymbol{w})$ can then be represented by a normal distribution with mean μ_w and covariance Σ_w , generating samples $\boldsymbol{w}_k \sim \mathcal{N}(\mu_w, \Sigma_w)$. Given the previous policy ***q(w)***, REPS obtains the new policy $\pi(\boldsymbol{w})$ by adding a KL divergence bound ϵ between the newly obtained policy and the previous one to the optimization of the expected reward. The bound on the KL divergence limits the variation on the new policy and prevents the algorithm from being too greedy, causing severe problems in certain robotics applications.

$$\pi^* = \underset{\pi}{\mathrm{argmax}} \int \pi(\boldsymbol{w}) \mathrm{R}(\boldsymbol{w}) d\boldsymbol{w} \tag{5.2}$$

$$\text{s.t.} \quad \epsilon \geqslant \mathrm{KL}(\pi(\boldsymbol{w})||\boldsymbol{q(w)}) \quad \text{and} \quad 1 = \textstyle\int \pi(\boldsymbol{w}) d(\boldsymbol{w})$$

where $\boldsymbol{w}$ are the parameters, $\mathrm{R}(\boldsymbol{w})$ is their associated reward (it is always negative, closer to 0 better is the result of the rollout examined), $\pi(\boldsymbol{w})$ and $q(\boldsymbol{w})$ the new and previous policies, respectively.

The constrained optimization problem (5.2) can be solved efficiently by the method of Lagrangian multipliers. From the Lagrangian, we can also obtain a closed-form solution for the new policy:

$$\pi^* \quad \propto \quad q(\boldsymbol{w}) \cdot e^{\frac{\mathrm{R}(\boldsymbol{w})}{\eta}} \tag{5.3}$$

where η is the Lagrange multiplier connected to the KL-bound constrain in the optimization problem (5.2). The parameter η is obtained by minimizing the dual function $g(\eta)$ of the original optimization problem (5.4). Details about derivation of the dual function used in the original REPS article [17] can be found in Appendix A.

$$g(\eta) = \eta\epsilon + \eta \log \int q(\boldsymbol{w}) e^{\frac{\mathrm{R}(\boldsymbol{w})}{\eta}} d\boldsymbol{w} \tag{5.4}$$

In practice, the integral in the dual function is approximated by a sum over *K* samples.

$$g(\eta) = \eta\epsilon + \eta \log \left[\frac{1}{K} \sum_{k=1}^{K} e^{\frac{\mathrm{R}(\boldsymbol{w})_k}{\eta}} \right] \tag{5.5}$$

It is common practice to introduce a normalization over rewards value with respect to $\mathrm{R}_{max} := \max_k(\mathrm{R}(\boldsymbol{w}_k))$ in order to avoids numerical issue during the learning process. Thus, the dual function (5.5) is re-written as (5.6).

$$g(\eta) = \eta\epsilon + \eta \log \left[\frac{1}{K} \sum_{k=1}^{K} e^{\frac{\mathrm{R}(\boldsymbol{w})_k - \mathrm{R}_{max}}{\eta}} \right] + \mathrm{R}_{max} \tag{5.6}$$

Given the value of η and the rewards, the exponential term in (5.3) acts as a weight to be used with the samples $\boldsymbol{w}_k$ to obtain the new policy with a *Gaussian Weighted Maximum Likelihood Estimation*.

REPS has the advantage to compute the optimal value for the η at each learning iteration. Therefore, this algorithm has only one open parameter, the KL bound ϵ, but its choice is not so critical and a value of 0.5 works well in most of the cases. REPS is afflicted by the problem of premature convergence of the covariance matrix (like many other policy search algorithms using maximum likelihood estimation), but this effect can be mitigated adding a regularization term (decreasing with the learning iterations) to the update of the covariance matrix. This term takes the form of $\lambda \cdot 0.99^i \cdot I_{3\times 3}$ (where *i* indicates the number of updates executed) so that it is bigger in the early learning stages, when the aim is to explore more and avoid converging prematurely, and smaller at the end of the learning process, when it is only needed to refine the trajectory, and adding too much variance when regularizing will make the algorithm explore far away from where we want to refine. The constant λ has to be tuned in each learning problem to ensure the more adequate speed of convergence for the covariance matrix. Application of REPS to ProMP policies is described in Algorithm 4.

Algorithm 4: Relative Entropy Policy Search for ProMP policies

Data: μ_w, Σ_w initial *ProMP* model

Result: μ_w^{new}, Σ_w^{new} updated *ProMP* model

for *k=1...K* **do**

  sample $\boldsymbol{w}_k \sim \mathcal{N}(\mu_w, \Sigma_w)$

  compute the rollout $\boldsymbol{\tau}_k = \boldsymbol{\Phi}_t^T \boldsymbol{w}_k$

repeat

  evaluate rewards $\mathrm{R}_k := \mathrm{R}(\boldsymbol{w}_k)$ for *k=1...K*

  $\mathrm{R}_{max} := \max_k(\mathrm{R}_k)$

  $\eta^* = \underset{\eta}{\operatorname{argmin}}\ \eta\epsilon + \eta \log\left[\frac{1}{K}\sum_k e^{(\mathrm{R}_k - \mathrm{R}_{max})/\eta}\right] + \mathrm{R}_{max}$

  $d_{\boldsymbol{w}}^k = e^{(\mathrm{R}_k - \mathrm{R}_{max})/\eta^*}$

  $\mu_{\boldsymbol{w}}^{new} = \sum_{k=1}^{K} d_{\boldsymbol{w}}^k\ \boldsymbol{w}_k$

  $\Sigma_{\boldsymbol{w}}^{new} = \frac{1}{K-1}\sum_{k=1}^{K} d_{\boldsymbol{w}}^k (\boldsymbol{w}_k - \mu_w)(\boldsymbol{w}_k - \mu_w)^T + \lambda \cdot 0.99^i \cdot I_{3\times 3}$;

until *convergence of* R_k;

5.2 Learning of symmetric bimanual tasks

REPS is the algorithm that has been adopted to perform the policy search for symmetric bimanual tasks. The traditional way of proceeding would be to model the movements of two robot-arms independently one from the other with two ProMPs subject to two separated learning processes.

Employing the symmetrization techniques presented in this work it is possible to develop a new learning approach, capable of exploiting the symmetric nature of a task. The objective is to reduce in this way the dimensionality of the learning problem and possibly increase its speed.

Before starting the learning process, some demonstrations of the task are recorded to build initial ProMP$_1$, $\boldsymbol{w}^1 \sim \mathcal{N}(\mu_{\boldsymbol{w}}^1, \Sigma_{\boldsymbol{w}}^1)$, modeling movements of the first robot's end-effector, and ProMP$_2$, $\boldsymbol{w}^2 \sim \mathcal{N}(\mu_{\boldsymbol{w}}^2, \Sigma_{\boldsymbol{w}}^2)$, modeling movements of the second robot's end-effector. Then, from demonstrations symmetry surface ρ^0 (indicating generally any plane, sphere or cylinder defining a symmetry in the movements) is estimated. At this point the learning process can start, following one of the approaches proposed.

- **Double Learning**: *Learn ProMP$_1$ and ProMP$_2$*
  The two ProMPs derived from demonstrations are updated both in the learning process. In this way the inner symmetry of the bimanual task is not taken into account, and there is no guarantee that the movements generated by the updated ProMPs are symmetric. Here REPS works on the probability distribution of the composed weight vector $\begin{bmatrix}\boldsymbol{w}_1\\ \boldsymbol{w}_2\end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix}\mu_{\boldsymbol{w}}^1\\ \mu_{\boldsymbol{w}}^2\end{bmatrix}, \begin{bmatrix}\Sigma_{\boldsymbol{w}}^1 & *\\ * & \Sigma_{\boldsymbol{w}}^2\end{bmatrix}\right)$. The non-diagonal blocks are initialized with all zeros, but with the learning updates their values will change, capturing any possible correlation between the two set of weights.

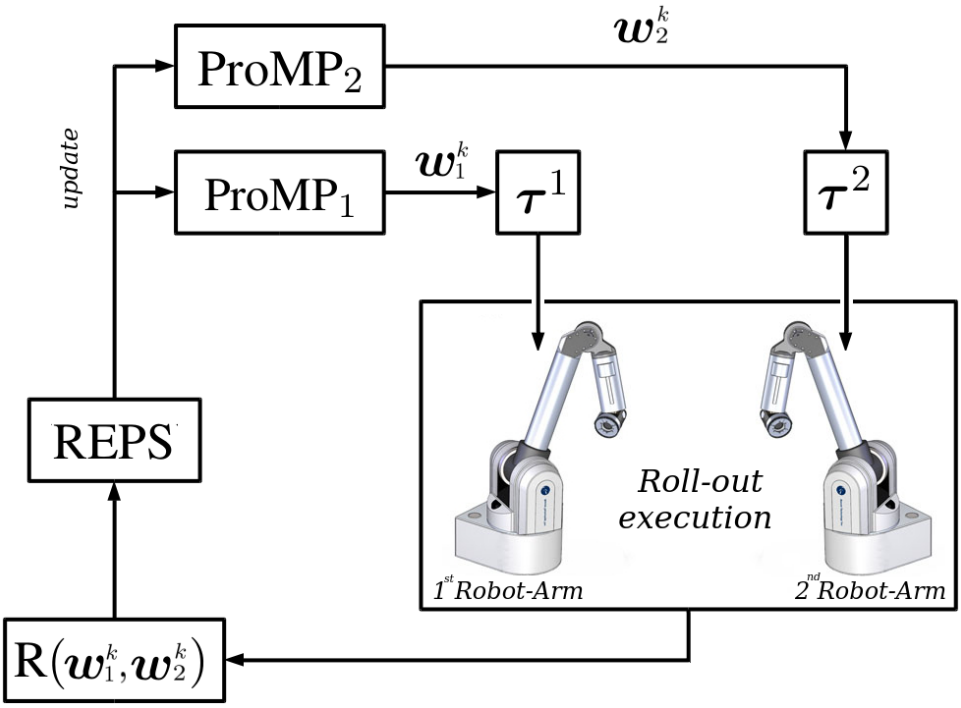


Figure 5.1: *Double Learning conceptual scheme.*

- **Symmetric Learning**: *Learn ProMP$_1$ + symmetrization*
  Only ProMP$_1$ is updated during the learning process. At each iteration the first arm executes the movement sampled from ProMP$_1$, while the second arm executes the symmetric trajectory. Once completed the policy search, movements of the first arm are described by the updated ProMP$_1$ and movements of the second by its symmetrization ProMP$_1^S$.

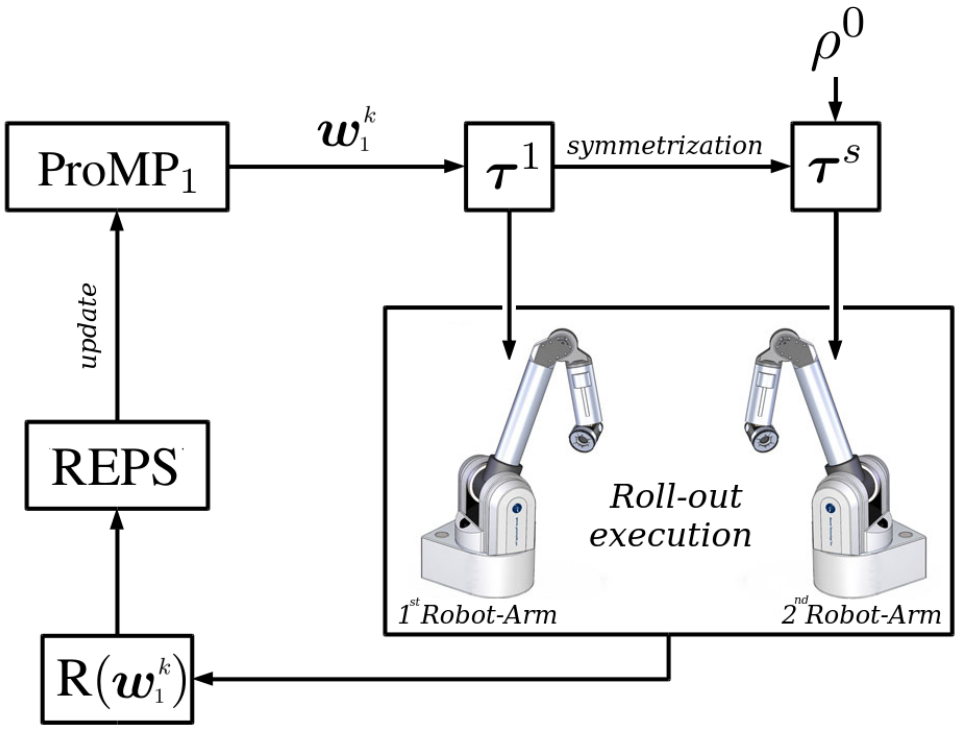


Figure 5.2: *Symmetric Learning conceptual scheme.*

Second approach reduces the amount of parameters to be updated to half, but its success depends critically on the accuracy of the symmetry surface estimation ρ^0. Surface obtained from demonstrations can be in fact wrong, due to poor execution of movements by the user, wrong measurements of relative positions between robots, or because the symmetry described by demonstrations is not exactly the one needed in the task execution. If it is not possible to achieve a good estimation **Symmetric Learning** is not reliable, and countermeasures need to be found.

The proposed solution is to learn, not only ProMP$_1$, but also symmetry surface's parameters, that can be represented as a Gaussian probability distribution, $\rho \sim \mathcal{N}(\rho^0, \Sigma_\rho)$. The value of covariance matrix Σ_ρ is obtained from (4.3-4.5-4.7).

- **Robust Symmetric Learning** *Learn ProMP$_1$ and surface + symmetrization*
  It is an extension of the second approach, that updates also ρ's parameters together with ProMP$_1$. In this way the policy search will involve also optimizing the symmetry surface so that the bimanual task is better executed. At each learning iteration k a trajectory for the first arm is symmetrized with respect to the sampled surface ρ^k, and then executed. The probability distribution used in REPS is defined as $\begin{bmatrix} \boldsymbol{w}_1 \\ \rho \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} \mu_{\boldsymbol{w}_1} \\ \rho^0 \end{bmatrix}, \begin{bmatrix} \Sigma_{\boldsymbol{w}_1} & * \\ * & \Sigma_\rho \end{bmatrix}\right)$, where weights and surface's parameters are combined in the same vector. Like in the first approach, non-diagonal blocks are initialized with zeros, and their values are updated during the process, capturing correlation between the weights and parameters of the symmetry surface.

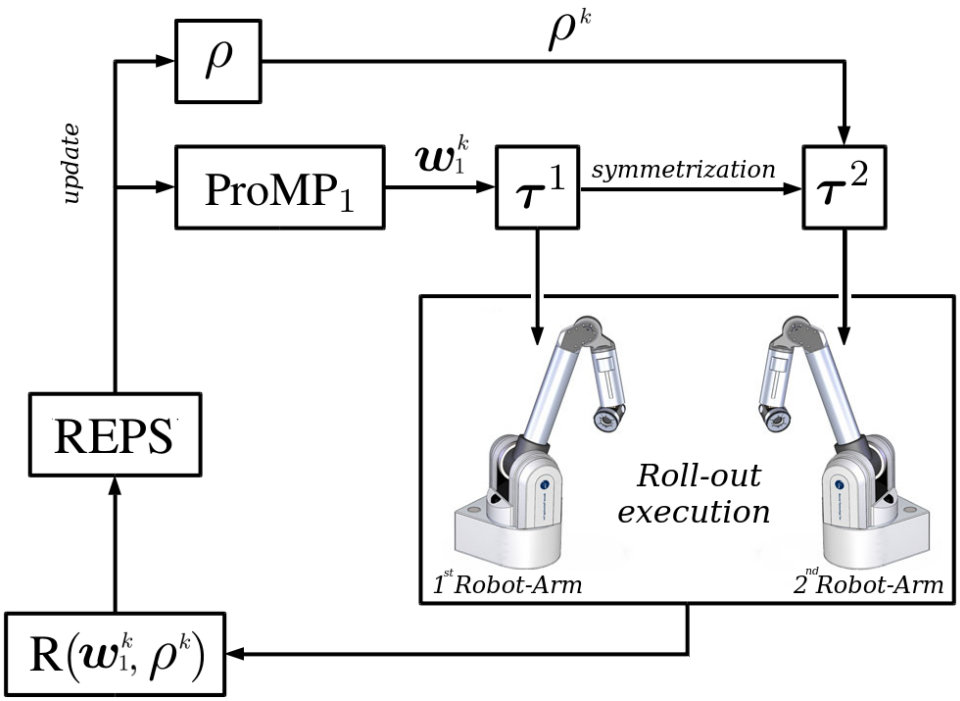


Figure 5.3: *Robust Symmetric Learning conceptual scheme.*

5.3 Testing in simulation

The proposed strategies, Double Learning (DL) and Symmetric Learning (SL), with its variant Robust Symmetric Learning (RSL), have been tested with some simple kinematic tasks, in order to verify their capacity to handle the learning of bimanual symmetric movements. For each test, a set of demonstrated movements has been recorded moving the robot-arms held in a gravity compensation state. These data have been used in MATLAB, where the whole learning process takes place, for simplicity. In fact it is possible to evaluate rewards for the sampled rollouts without the necessity of actually executing the movements with real robots. After that it is always possible to command robot-arms with the resulting updated ProMPs.

5.3.1 Test 1: Passage through via-points

The task requested is to pass through a set of via-points with the two end-effectors $(x^i_{VP}(k), y^i_{VP}(k), z^i_{VP}(k))$ at given common time instants t_k, for $k = 1 \dots 6$ and $i = 1, 2$. For each robot a set of six points is given, defined in such a way that a symmetry plane can be established between via-points assigned to one arm and via-points assigned to the other. Two different situations are taken into account: in the first case via-points symmetry plane is chosen equal to the estimated one π^0 (by method described in Chapter 4), while in the second case via-points are calculated with a plane π^{VP} whose parameters are obtained perturbing the ones of π^0 with a uniformly distributed random noise $\mathcal{U}(0, 0.2)$.

Movement policies have been represented with ProMPs built using $M = 10$ basis functions for each direction (i.e., $3 \cdot M = 30$ weights in total) and amplitude parameter $h = 5 \cdot 10^{-4}$ (see Chapter 2 for more details about structure of basis functions). The regularization term coefficient used in REPS that guarantees a right convergence speed for the learning of this task is $\lambda = 10^{-4}$.
The reward function considered (5.7, where i indicates the first or second arm, and k is the via-point index) penalizes the sum of squared errors in the passage through via-points.

$$\mathrm{R} = 100 \sum_{i=1}^{2} \sum_{k=1}^{6} \left[x^i_{VP}(k) - x^i_{t_k}\right]^2 + \left[y^i_{VP}(k) - y^i_{t_k}\right]^2 + \left[z^i_{VP}(k) - z^i_{t_k}\right]^2 \tag{5.7}$$

Known symmetry's parameters

The two approaches are compared initially in the simplest case, when the symmetry plane defining via-points is perfectly known. Both ways of proceeding succeed in learning the task, managing to make the end-effectors go through the desired via-points. By construction, SL generates symmetric movements for the two arms, while DL cannot assure the symmetry, but in the context of this precise task it is not going to be relevant. In Fig. 5.4-5.5 are shown the resulting trajectories for the updated ProMPs after the learning process ended. (Note how all the rollouts have converged to the mean trajectory for both arms, implying the presence of no variability in weight distribution after many policy updates, i.e., $\Sigma_{\boldsymbol{w}}$ for each ProMP has all eigenvalues almost equal to zero).

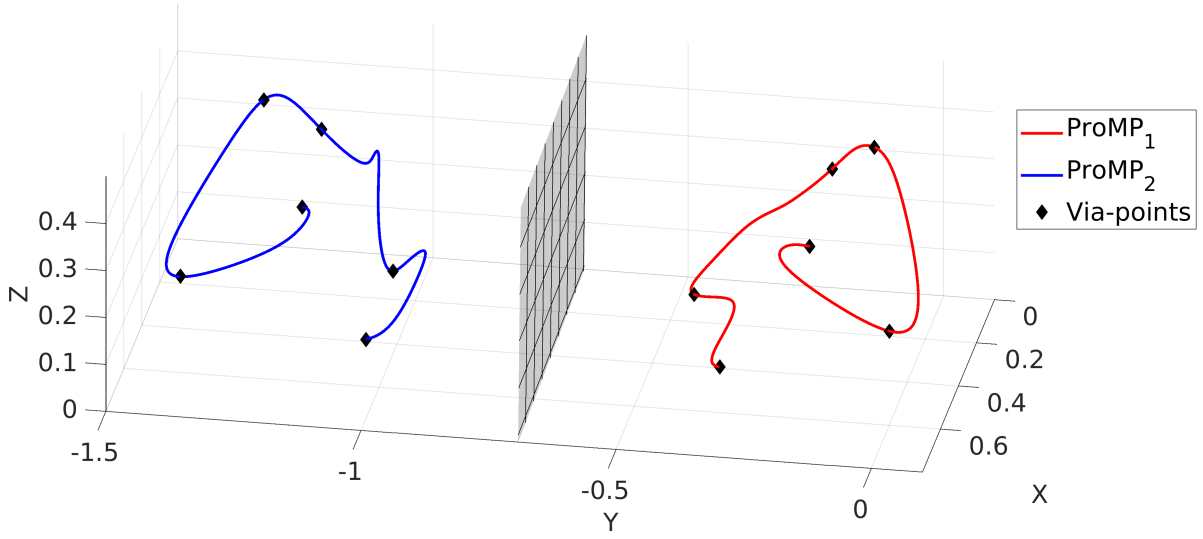


Figure 5.4: *Mean trajectories and rollouts for $ProMP_1$ and $ProMP_2$ obtained with DL applied to Test 1 with a known task symmetry plane.*

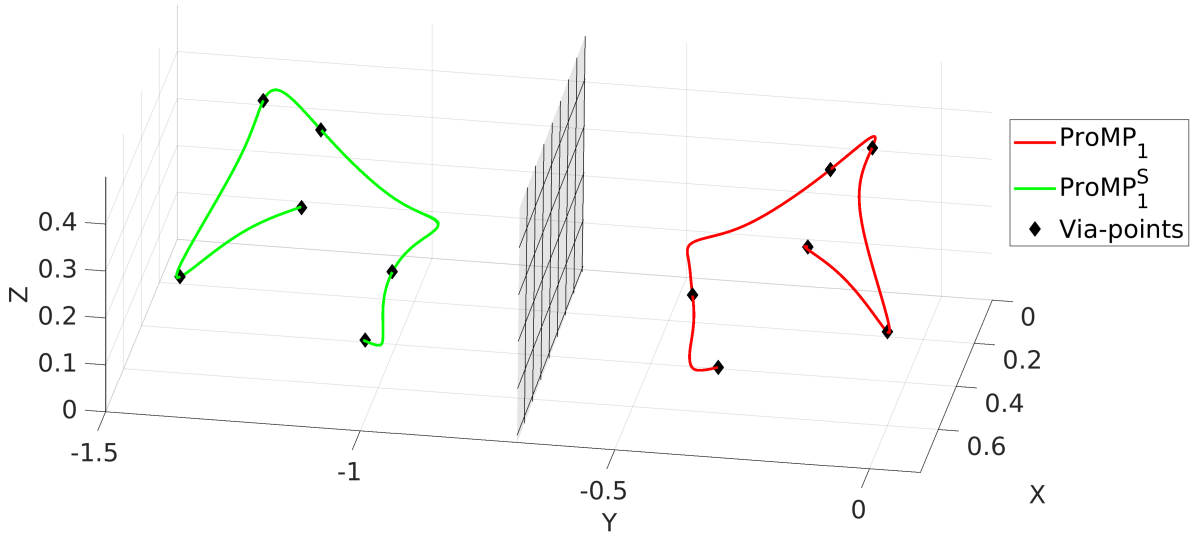


Figure 5.5: *Mean trajectories and rollouts for $ProMP_1$ and its symmetric $ProMP_1^S$ obtained with SL applied to Test 1 with a known task symmetry plane.*

Performance of the two strategies has been evaluated comparing the convergence speed of rewards during the learning process. Reward given by the mean trajectory has been measured $\mathrm{R}(\mu_{\boldsymbol{w}})$. The test has been repeated 15 times, reporting the average value $\bar{\mathrm{R}}$ for each update together with a 95% confidence interval, obtained as $\left(\bar{\mathrm{R}} - 1.96\frac{\sigma}{\sqrt{15}}, \bar{\mathrm{R}} + 1.96\frac{\sigma}{\sqrt{15}}\right)$ with σ indicating the standard deviation of the reward samples. Results of the analysis are reported in Table 5.1 and Fig. 5.6.

Table 5.1: *$R(\mu_{\boldsymbol{w}})$ evolution in Test 1 with known symmetry's parameters.*

	START	*10 updates*	*50 updates*	*100 updates*	*200 updates*	*500 updates*
DL	-87.1	-54.4 $\pm$ 4.2	-16.2 $\pm$ 2.2	-1.8 $\pm$ 0.34	-0.049 $\pm$ 0.004	-0.0084 $\pm$ 0.0003
SL	-87.4	-43.9 $\pm$ 5.4	-5.2 $\pm$ 2.1	-0.12 $\pm$ 0.04	-0.033 $\pm$ 0.006	-0.0073 $\pm$ 0.0002

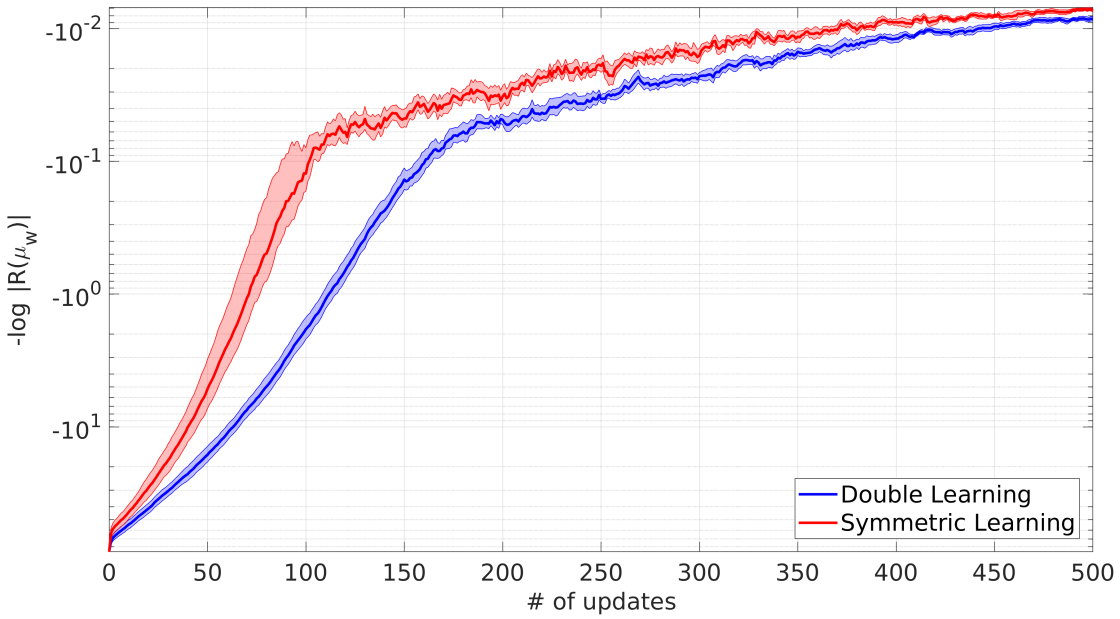


Figure 5.6: *$R(\mu_{\boldsymbol{w}})$ convergence in Test 1 with known task symmetry plane (values in logarithmic scale).*

SL shows a faster convergence in the policy search, mostly in the early phases of the learning process (roughly the first 100 updates), exactly the kind of results sought by the application of symmetrization techniques in the movement policy representation. Although, this procedure depends critically on the accuracy of the symmetry plane estimated. If the parameters of the plane defining the task are exactly known, ProMP_1 is updated correctly and its execution together with the symmetric ProMP_1^S manage to execute the task. But, in case the parameters of symmetry plane are unknown, learning algorithm uses a wrong estimation to calculate the symmetrization, and the resulting ProMPs generates movements that fails in the execution of the task. In Fig. 5.7 an example of this situation happening in Test 1 is illustrated.

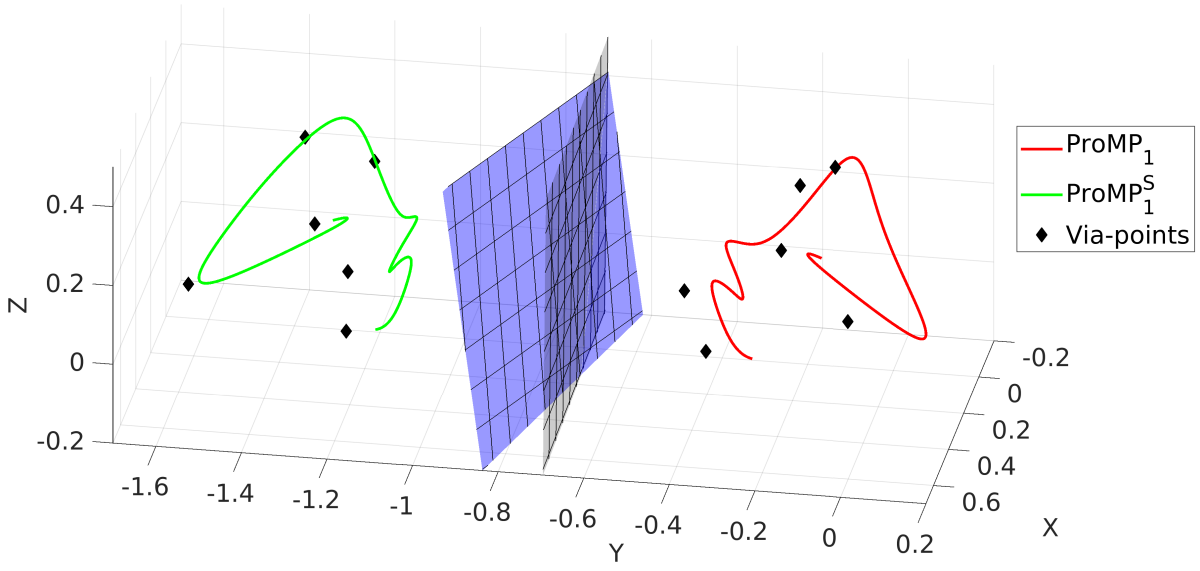


Figure 5.7: *Failure in learning to go through the desired via-points due to the uncertain knowledge of the task symmetry. In blue is indicated the real symmetry plane, different from the estimation known by the algorithm.*

Unknown symmetry's parameters

RSL, instead, is reliable also in situations where it is not possible to know precisely the task symmetry. This strategy starts using the plane estimate π^0 derived from initial demonstrations, and then updates its parameters at each iteration π^k trying to find the surface that allows to execute better the task. In Test 1 this way of proceeding is successful and manages to handle the case of via-points obtained from a unknown symmetrization. At each update the learned plane π^k get closer to the real one, converging to it almost exactly after some iterations. Resulting trajectories are shown in Fig. 5.8.

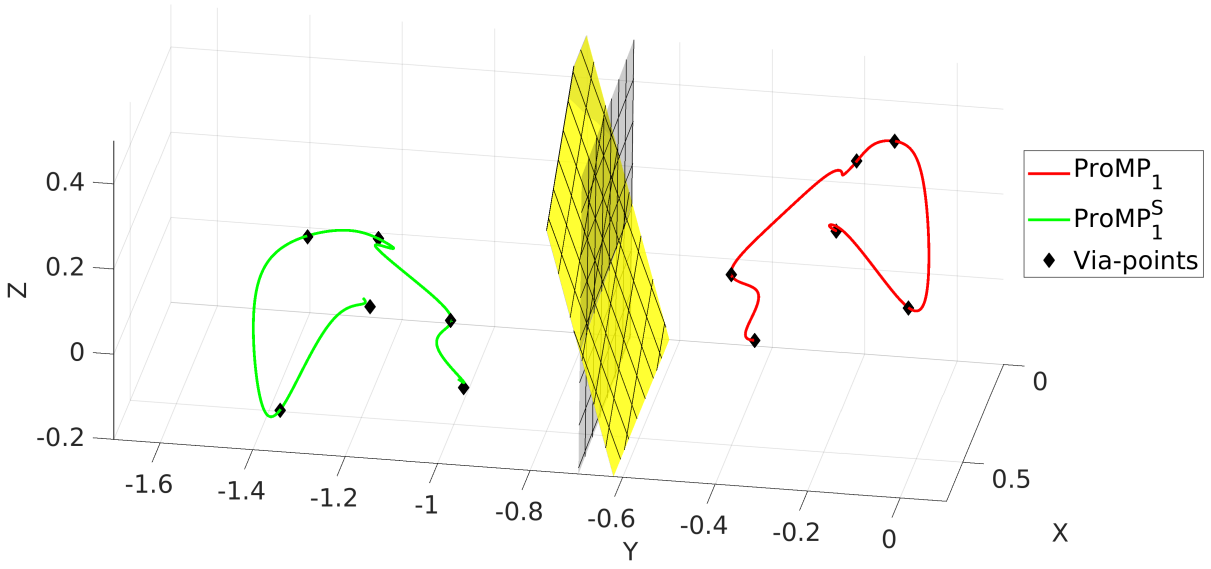


Figure 5.8: *Mean trajectories and rollouts for ProMP$_1$ and its symmetric ProMP$_1^S$ obtained with RSL applied to Test 1 with an unknown task symmetry plane. The learned plane (plotted in yellow) approximates accurately the real symmetry of the task.*

On the other hand, DL does not take into account the presence of symmetry in the movements of end-effectors to learn the task, so it is not influenced by the presence of eventual uncertainties in the symmetry plane estimated.

Also in this case the test has been repeated 15 times employing both learning approaches to study the convergence of R($\mu_{\boldsymbol{w}}$). In each repetition the same perturbed plane's parameters have been considered with both approaches in order to make a significant comparison between the two. Results are reported in Table (5.2) and Fig. (5.9).

Table 5.2: *$R(\mu_{\boldsymbol{w}})$ values in Test 1 with unknown symmetry's parameters.*

	START	*10 updates*	*50 updates*	*100 updates*	*200 updates*	*500 updates*
DL	-113.0 $\pm$ 6.7	-68.4 $\pm$ 5.7	-24.3 $\pm$ 3.4	-3.6 $\pm$ 0.9	-0.055 $\pm$ 0.007	-0.0086 $\pm$ 0.0003
RSL	-107.1 $\pm$ 6.5	-63.3 $\pm$ 8.7	-11.2 $\pm$ 2.5	-1.1 $\pm$ 0.3	-0.073 $\pm$ 0.012	-0.0095 $\pm$ 0.0005

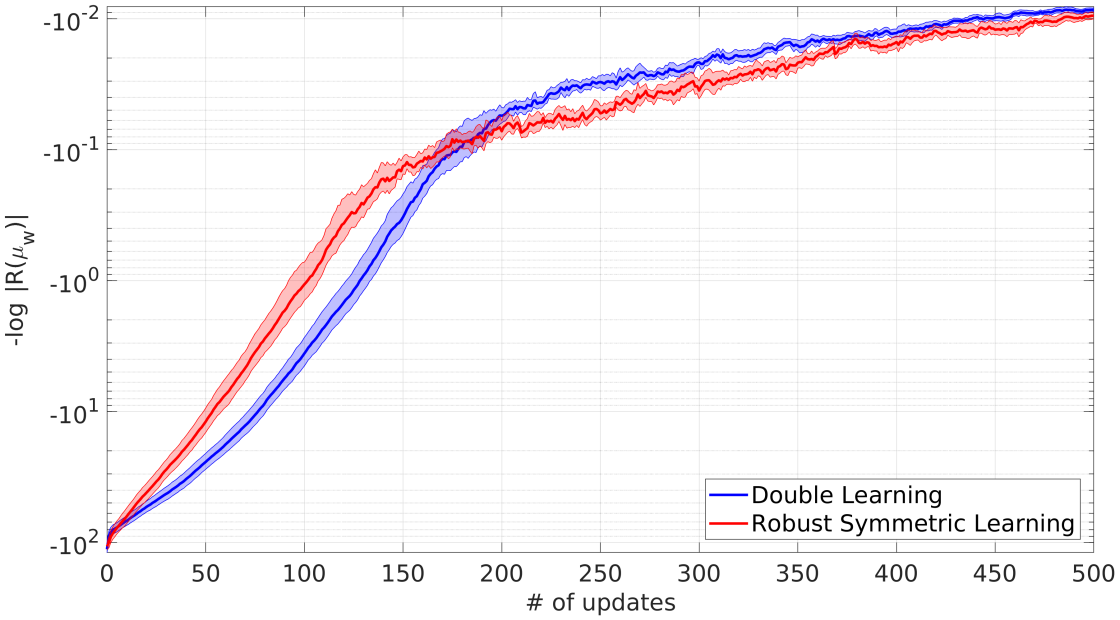


Figure 5.9: $R(\mu_{\boldsymbol{w}})$ *convergence in Test 1 with unknown task symmetry plane (values in logarithmic scale).*

RSL shows a slower convergence than SL in the previous case, when the plane was known. This is due to the fact that now it is necessary to learn the plane's parameters too, together with the movement policy. With respect to DL its convergence is still faster at the beginning, while later, around 150 updates, it cannot manage to increase rewards as much as DL does. This is still a good result, because it can be seen that using symmetry in the learning, also in presence of uncertainties, can reach faster a good solution. The fact that later, with more updates, RSL looses effectiveness depends on the fact that using two separate ProMPs allows to optimize better the rewards, having more parameters at disposal.

5.3.2 Test 2: Follow a path with end-effectors' midpoint

In Test 2 it is asked to the midpoint between two robot end-effectors at each time step $M_t = (x_t^M, y_t^M, z_t^M)$, defined as $M_t := \left(\frac{x_t^1+x_t^2}{2}, \frac{y_t^1+y_t^2}{2}, \frac{z_t^1+z_t^2}{2}\right)$ for $t = 1 \dots N_t$, to follow a certain path $\{x_t^{ref}, y_t^{ref}, z_t^{ref}\}_{t=1\dots N_t}$ belonging to a surface. Three different scenarios have been considered: reference path belonging to a plane, a sphere or a cylinder. This surface defines a symmetry in the movements, and, like before, two different situations are taken into account: midpoint path belongs to surface ρ^0 known by learning algorithm, or it belongs to a different surface obtained by perturbing ρ^0's parameters with a uniformly distributed random noise $\mathcal{U}(0, 0.1)$.
ProMPs have $M = 10$ basis functions for each direction (i.e., $3 \cdot M = 30$ weights in total) and amplitude parameter $h = 0.02$ (see Chapter 2 for more details about structure of basis functions). The regularization term coefficient used in REPS that guarantees a right convergence speed for the learning of this task is $\lambda = 10^{-5}$. The reward function considered (5.8) penalizes the sum of squared errors from the desired path on the surface.

$$\mathrm{R} = 100 \sum_{t=1}^{N_t} \left(x_t^{ref} - x_t^M\right)^2 + \left(y_t^{ref} - y_t^M\right)^2 + \left(z_t^{ref} - z_t^M\right)^2 \tag{5.8}$$

Known symmetry

When parameters defining the surface to which the reference path belongs are perfectly known from the estimation, DL and SL can be applied. The results for each kind of surface considered are described together. Both methods succeed in learning the task, drawing a midpoint path on ρ^0, presenting only minor deviations from the reference. From Fig. 5.10 to Fig. 5.15 resulting learned trajectories are shown.

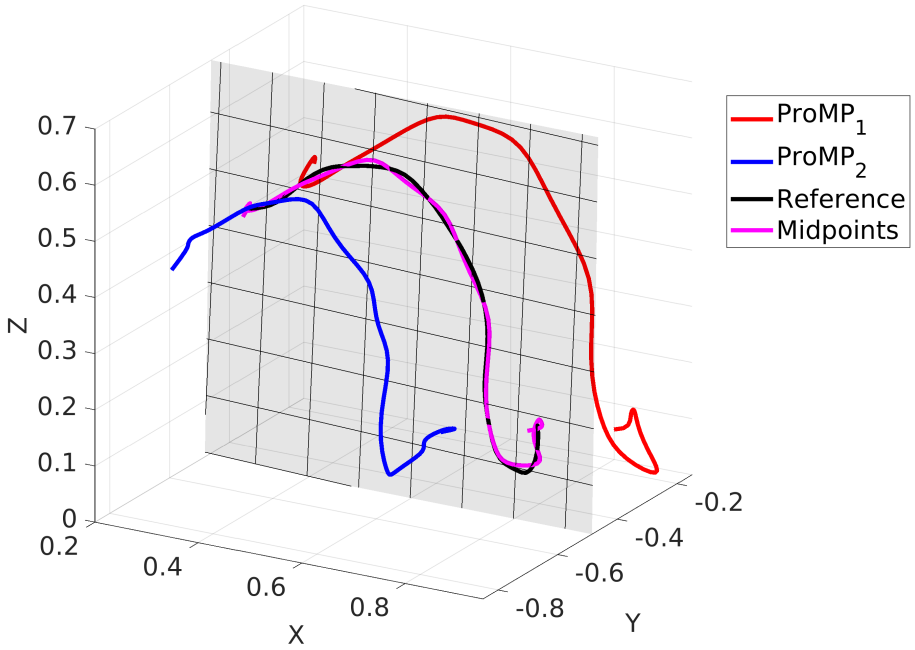


Figure 5.10: *Mean trajectories and rollouts for $ProMP_1$ and $ProMP_2$, obtained by DL applied to Test 2, in the case of reference path belonging to the estimated plane.*

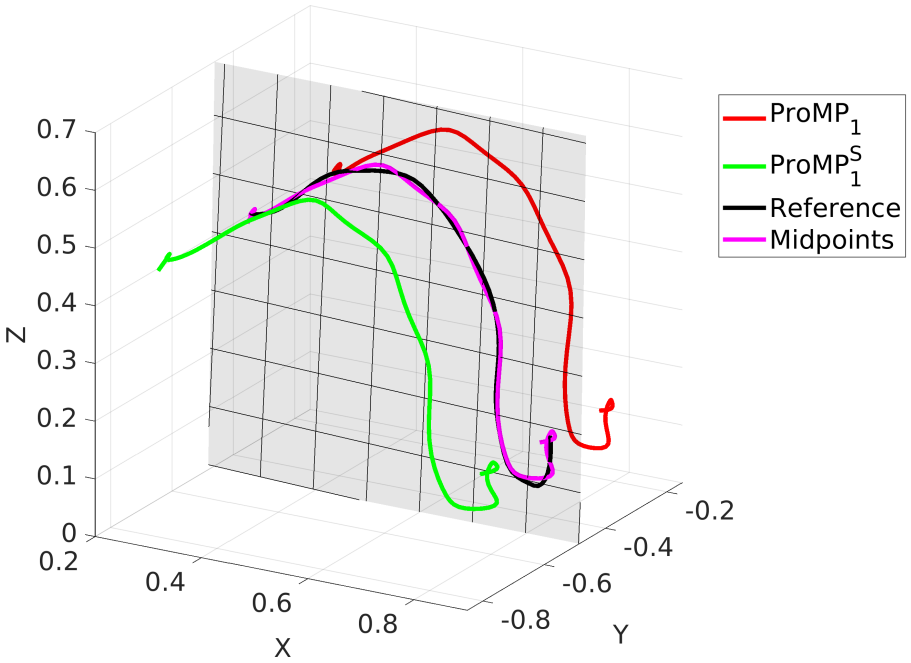


Figure 5.11: *Mean trajectories and rollouts for $ProMP_1$ and its symmetric $ProMP_1^S$, obtained by SL applied to Test 2, in the case of reference path belonging to the estimated plane.*

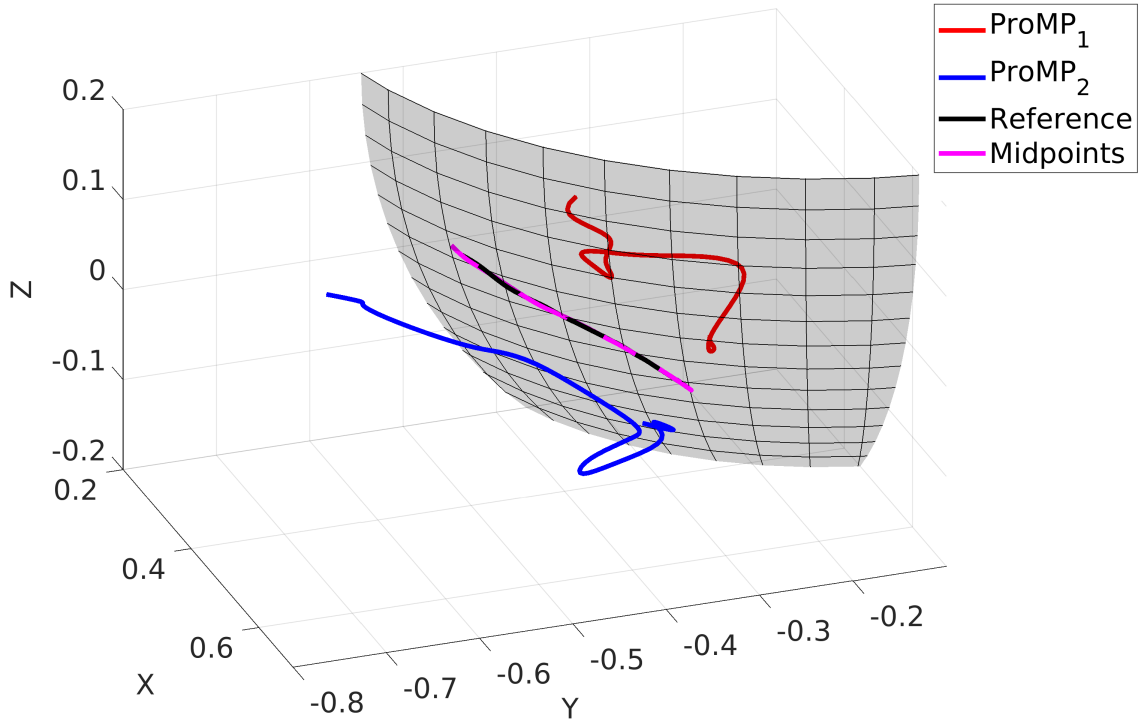


Figure 5.12: *Mean trajectories and rollouts for $ProMP_1$ and $ProMP_2$, obtained by DL applied to Test 2, in the case of reference path belonging to the estimated sphere.*

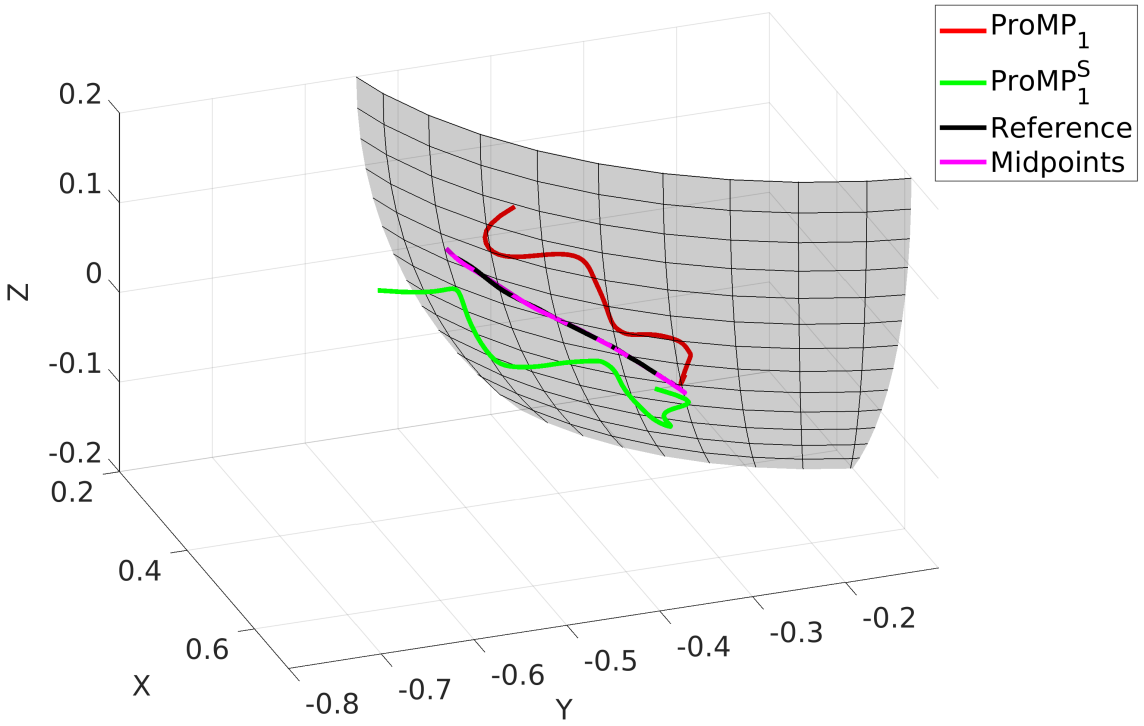


Figure 5.13: *Mean trajectories and rollouts for $ProMP_1$ and its symmetric $ProMP_1^S$, obtained by SL applied to Test 2, in the case of reference path belonging to the estimated sphere.*

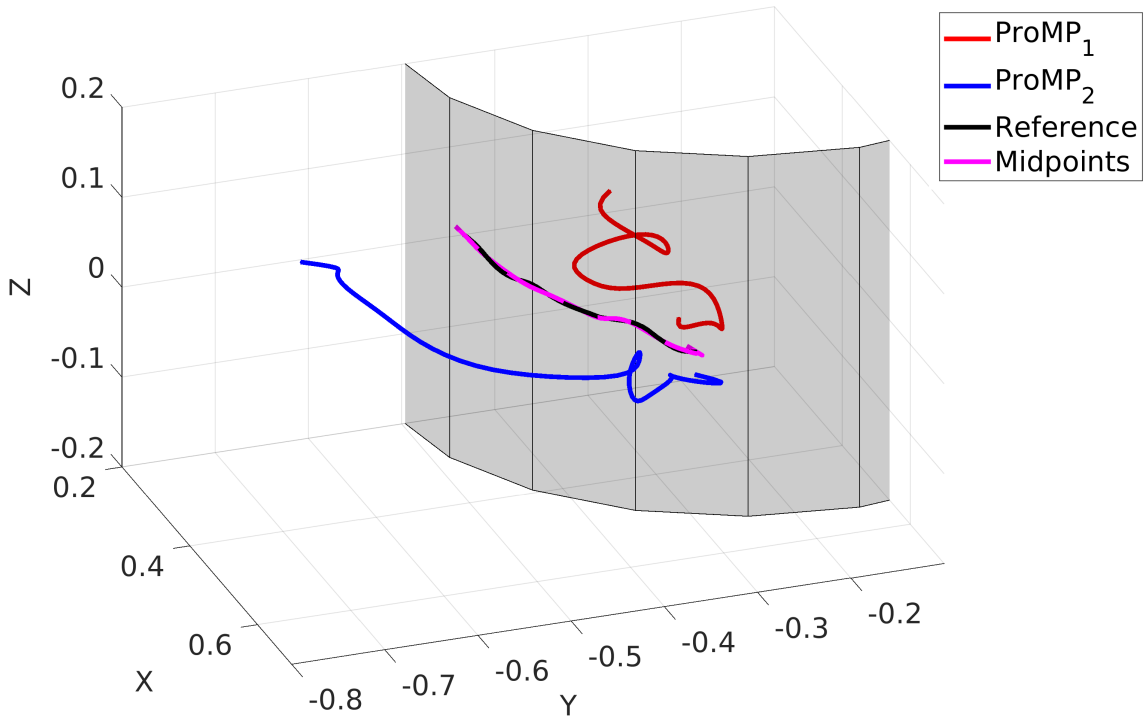


Figure 5.14: *Mean trajectories and rollouts for ProMP$_1$ and ProMP$_2$, obtained by DL applied to Test 2, in the case of reference path belonging to the estimated cylinder.*

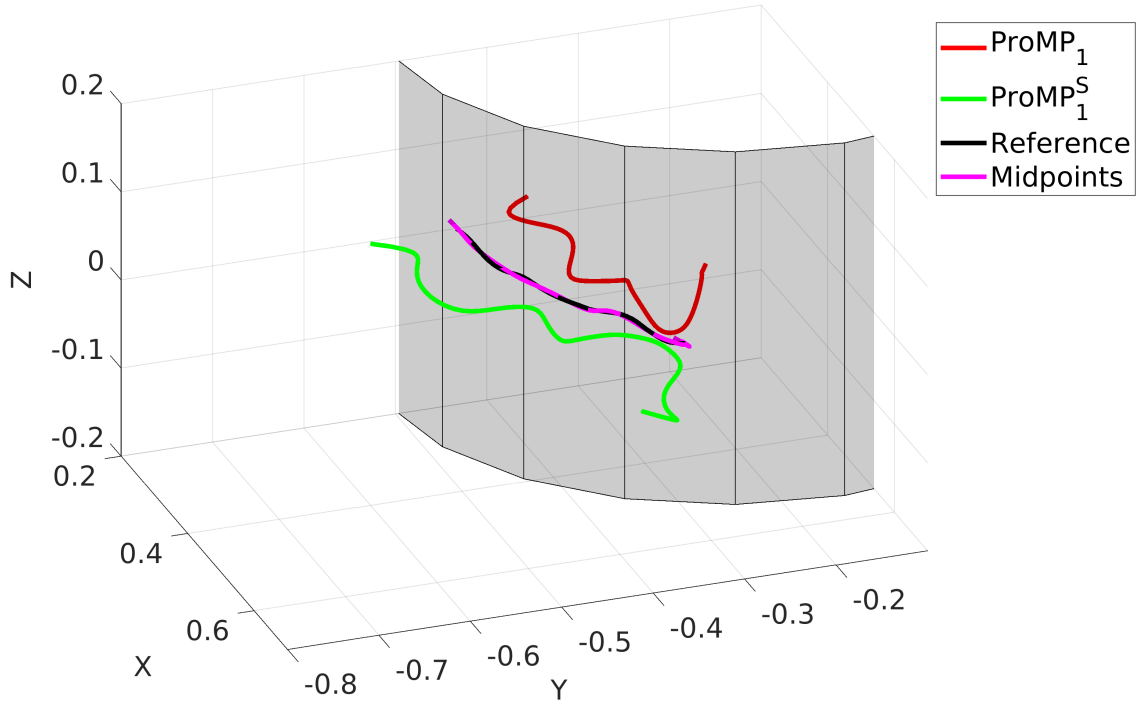


Figure 5.15: *Mean trajectories and rollouts for ProMP$_1$ and its symmetric ProMP$_1^S$, obtained by SL applied to Test 2, in the case of reference path belonging to the estimated cylinder.*

The rewards convergence has been evaluated for both strategies in each different scenarios. R($\mu_{\boldsymbol{w}}$) evolution has been measured in 15 repetitions of the test, reporting the average value $\bar{\text{R}}$ for each update together with a 95% confidence interval (obtained as described in Test 1). Also in these scenarios a faster

convergence of SL can be observed, regardless of the nature of the symmetry surface. Results of the convergence analysis are given from Fig. 5.16 to Fig. 5.18, from Table 5.3 to Table 5.5.

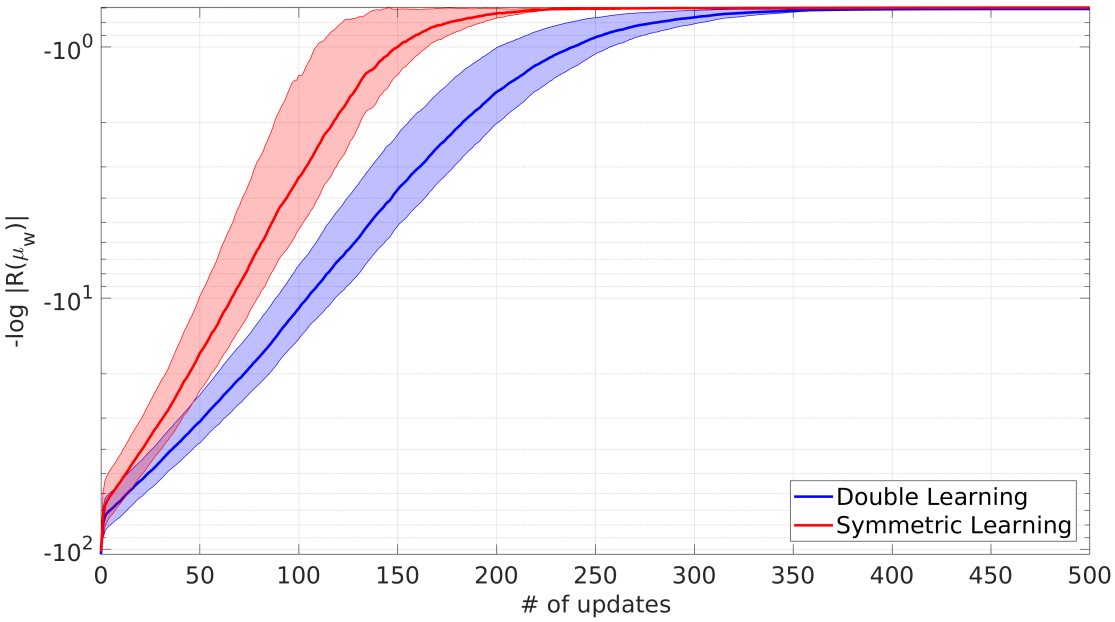


Figure 5.16: *$R(\mu_{\boldsymbol{w}})$ convergence in Test 2, in the case of reference path belonging to the estimated plane (values in logarithmic scale).*

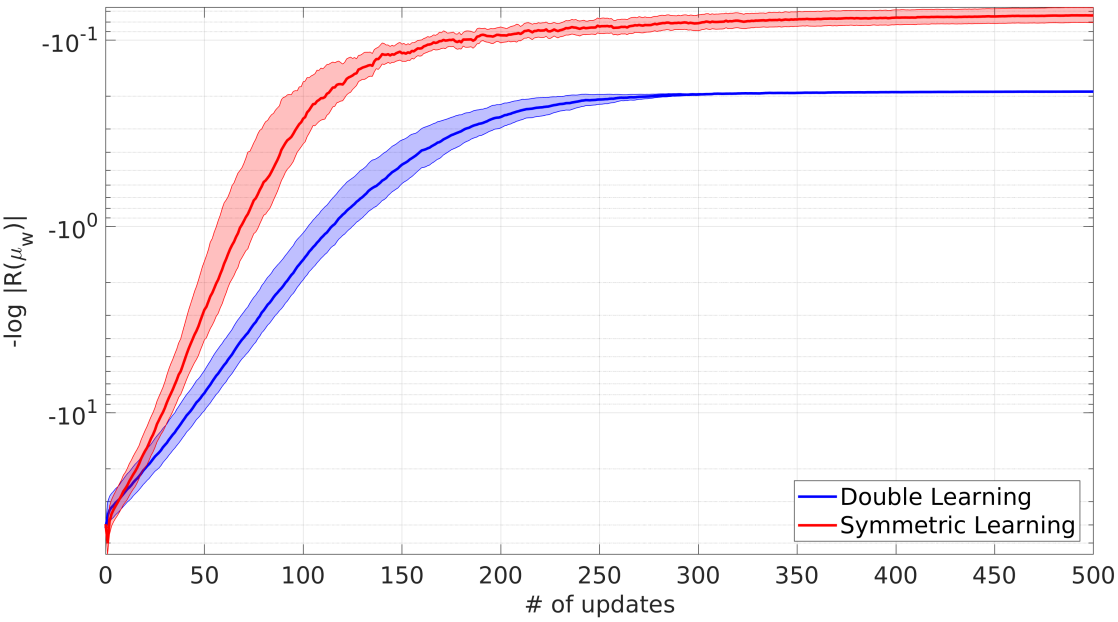


Figure 5.17: *$R(\mu_{\boldsymbol{w}})$ convergence in Test 2, in the case of reference path belonging to the estimated sphere (values in logarithmic scale).*

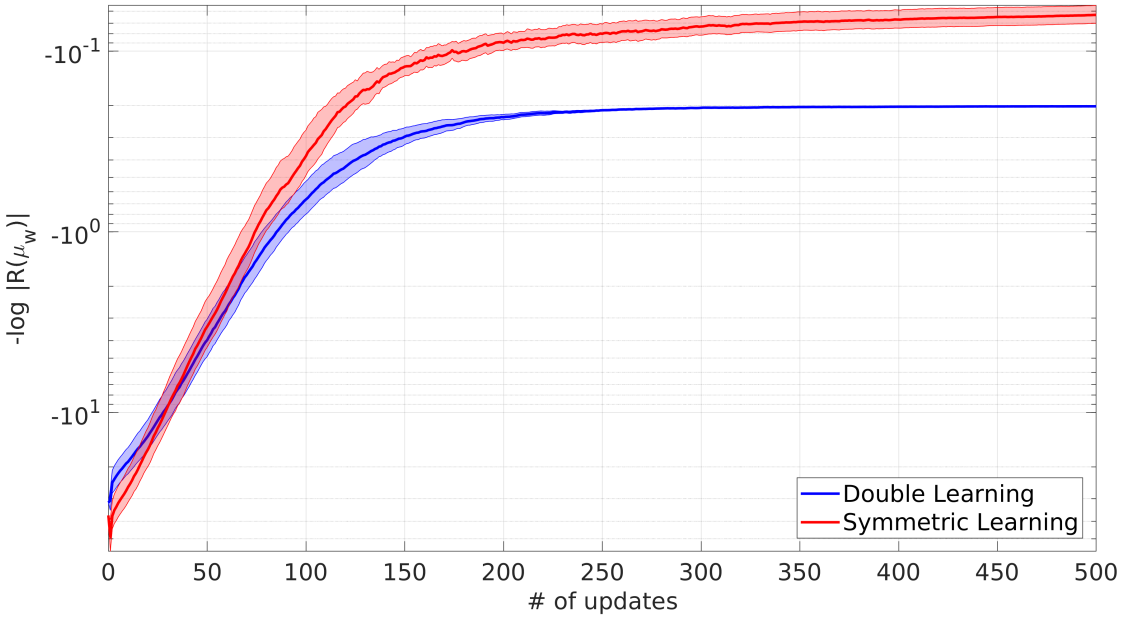


Figure 5.18: $R(\mu_{\boldsymbol{w}})$ *convergence in Test 2, in the case of reference path belonging to the estimated cylinder (values in logarithmic scale).*

Table 5.3: $R(\mu_{\boldsymbol{w}})$ *values in Test 2, for path belonging to a plane, with known symmetry's parameters.*

	START	*10 updates*	*50 updates*	*100 updates*	*200 updates*	*500 updates*
DL	-104.8	-63.9 $\pm$ 10.2	-31.0 $\pm$ 6.7	-11.0 $\pm$ 3.6	-1.5 $\pm$ 0.5	-0.7054 $\pm$ 0.0001
SL	-102.0	-53.7 $\pm$ 12.1	-16.6 $\pm$ 6.8	-3.3 $\pm$ 2.0	-0.73 $\pm$ 0.03	-0.6964 $\pm$ 0.0001

Table 5.4: $R(\mu_{\boldsymbol{w}})$ *values in Test 2, for path belonging to a sphere, with known symmetry's parameters.*

	START	*10 updates*	*50 updates*	*100 updates*	*200 updates*	*500 updates*
DL	-41.4	-26.5 $\pm$ 4.6	-7.8 $\pm$ 1.9	-1.51 $\pm$ 0.43	-0.259 $\pm$ 0.037	-0.1890 $\pm$ 0.0001
SL	-39.6	-25.5 $\pm$ 4.8	-2.8 $\pm$ 1.3	-0.26 $\pm$ 0.10	-0.094 $\pm$ 0.009	-0.0737 $\pm$ 0.0068

Table 5.5: $R(\mu_{\boldsymbol{w}})$ *values in Test 2, for path belonging to a cylinder, with known symmetry's parameters.*

	START	*10 updates*	*50 updates*	*100 updates*	*200 updates*	*500 updates*
DL	-31.5	-18.7 $\pm$ 3.1	-4.0 $\pm$ 0.9	-0.66 $\pm$ 0.14	-0.232 $\pm$ 0.008	-0.2018 $\pm$ 0.0001
SL	-37.1	-25.8 $\pm$ 5.6	-3.3 $\pm$ 1.0	-0.38 $\pm$ 0.11	-0.090 $\pm$ 0.009	-0.0630 $\pm$ 0.0071

Unknown symmetry

Test has been repeated for the case of a reference path belonging to an unknown surface (obtained by random perturbation of estimated ρ^0), to see if RSL can cope with uncertainties also in these scenarios. The obtained results are shown From Fig. 5.19 to Fig. 5.21.

RSL manages to learn a good approximation of the symmetry surface defining the task. Thanks to that the robots are able to learn executing correctly the task also in presence of uncertainties.

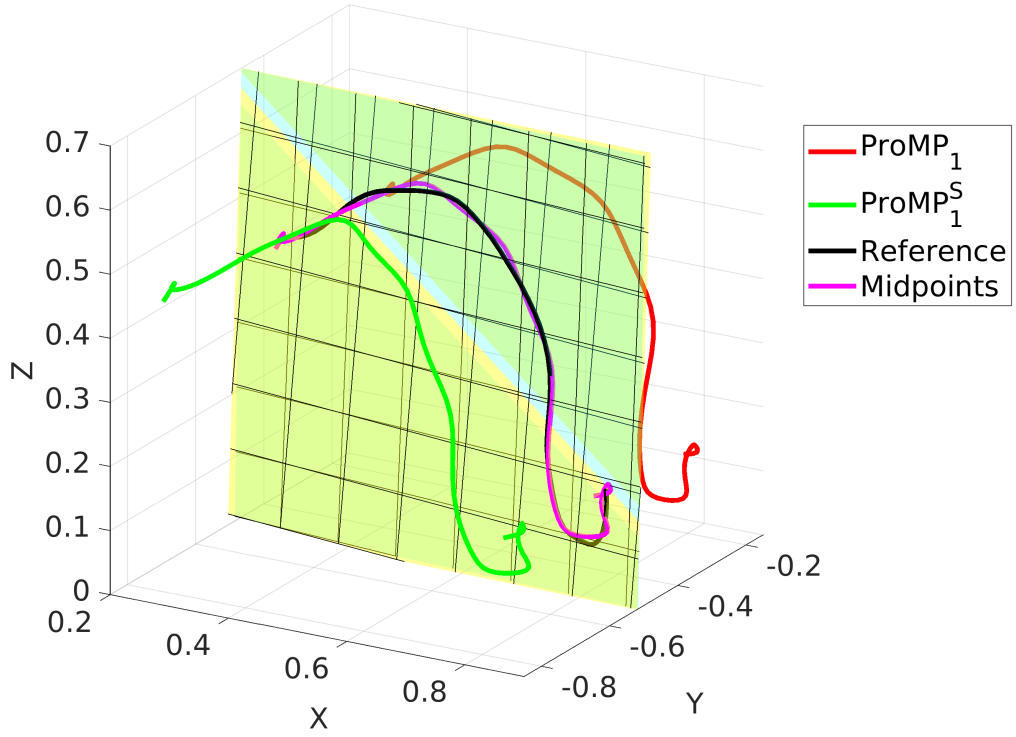


Figure 5.19: *Mean trajectories and rollouts for ProMP$_1$ and its symmetric ProMP$_1^S$, obtained by RSL applied to Test 2, in the case of reference path belonging to an unknown plane (plotted in light blue). The learned surface (plotted in yellow) approximates accurately the real symmetry of the task.*

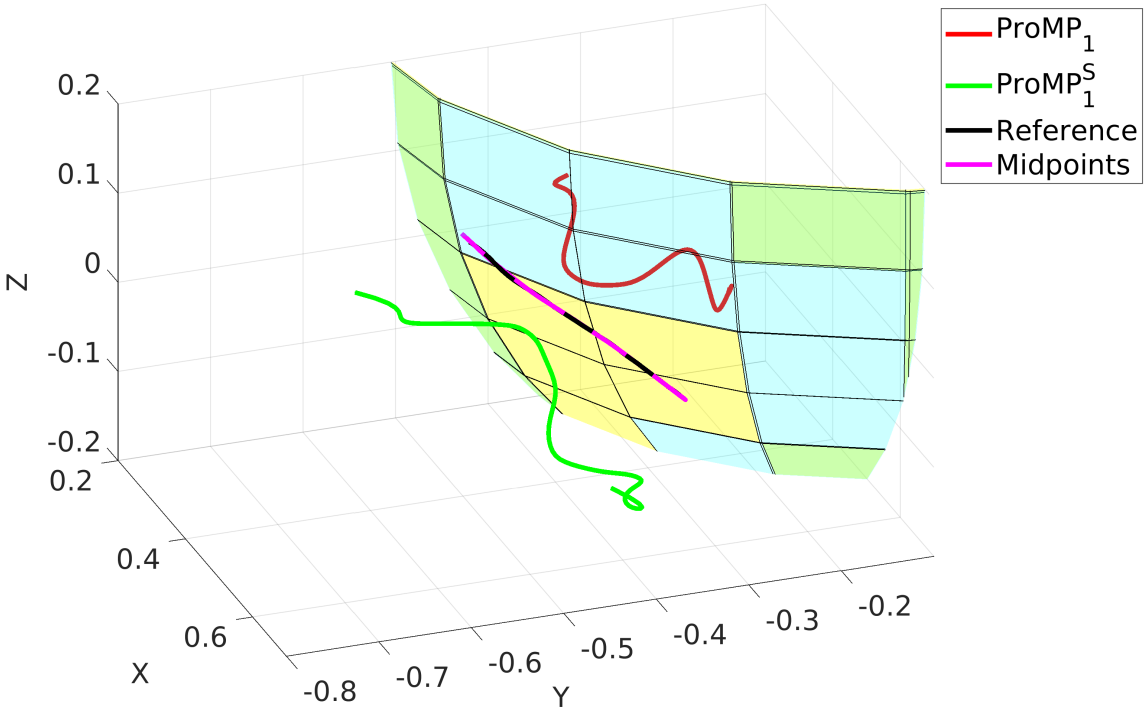


Figure 5.20: *Mean trajectories and rollouts for ProMP$_1$ and its symmetric ProMP$_1^S$, obtained by RSL applied to Test 2, in the case of reference path belonging to an unknown sphere (plotted in light blue). The learned surface (plotted in yellow) approximates accurately the real symmetry of the task.*

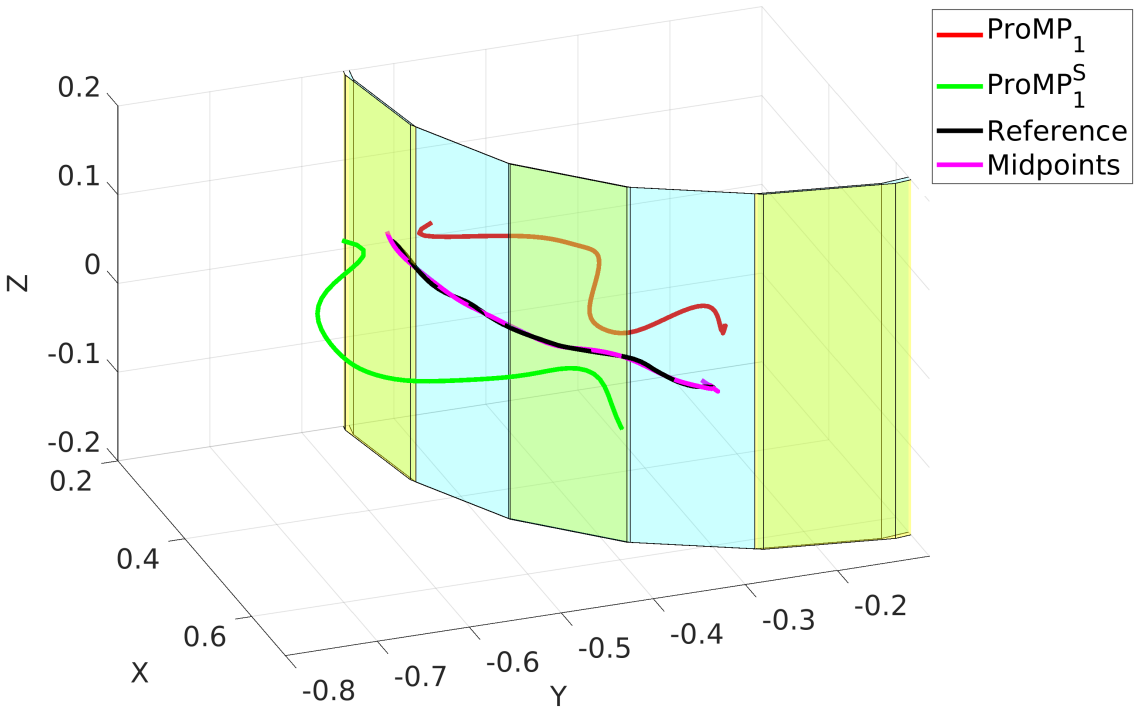


Figure 5.21: *Mean trajectories and rollouts for ProMP$_1$ and its symmetric ProMP$_1^S$, obtained by RSL applied to Test 2, in the case of reference path belonging to an unknown cylinder (plotted in light blue). The learned surface (plotted in yellow) approximates accurately the real symmetry of the task.*

R($\mu_{\boldsymbol{w}}$) convergence has been evaluated for both strategies in each different scenarios, repeating the test 15 times. Average value $\bar{\text{R}}$ for each update together with a 95% confidence interval are given (from Fig. 5.22 to Fig. 5.24, from Table 5.6 to Table 5.8. Dealing with unknown symmetry slow down the convergence of rewards, but RSL still manages to be faster of DL, mostly in the early phases of the learning process.

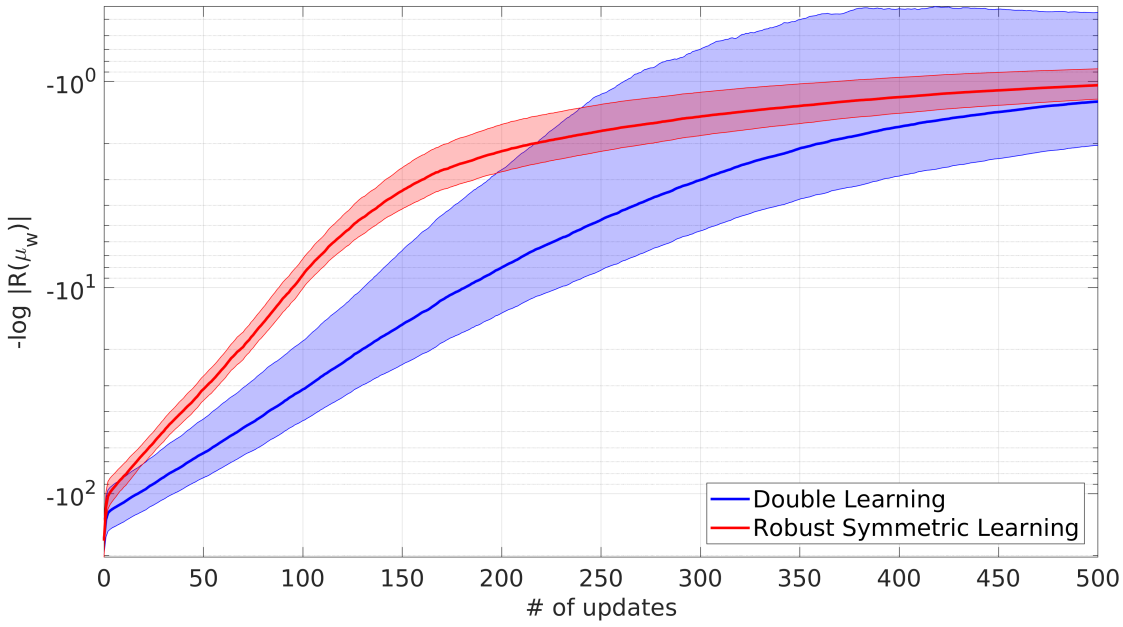


Figure 5.22: *R($\mu_{\boldsymbol{w}}$) convergence in Test 2, in the case of reference path belonging to an unknown plane (values in logarithmic scale).*

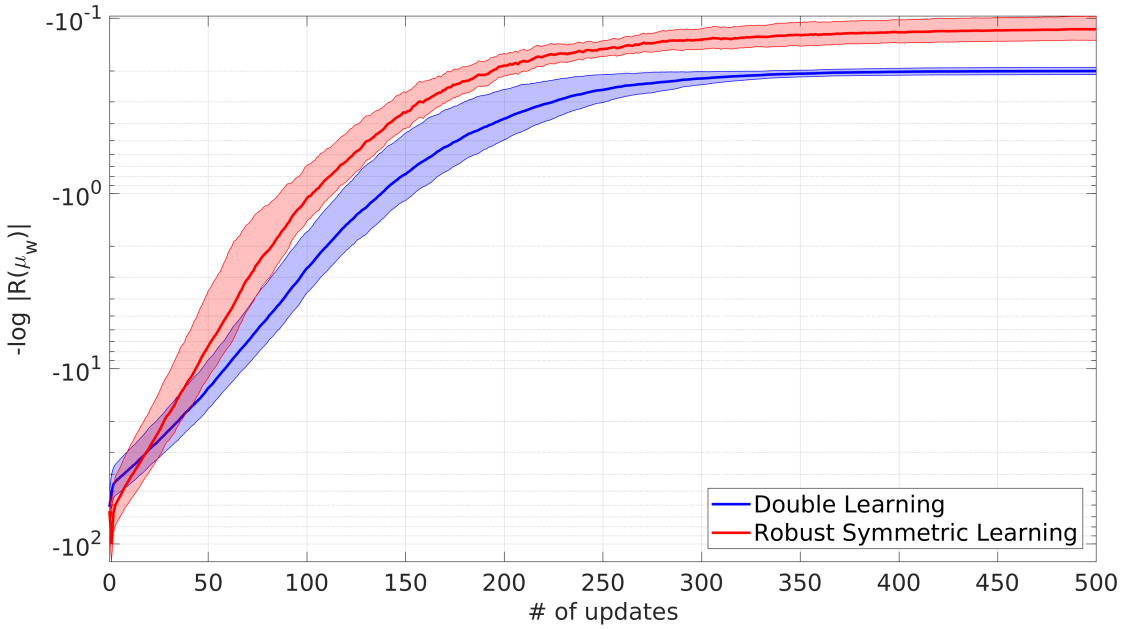


Figure 5.23: $R(\mu_{\boldsymbol{w}})$ *convergence in Test 2, in the case of reference path belonging to an unknown sphere (values in logarithmic scale).*

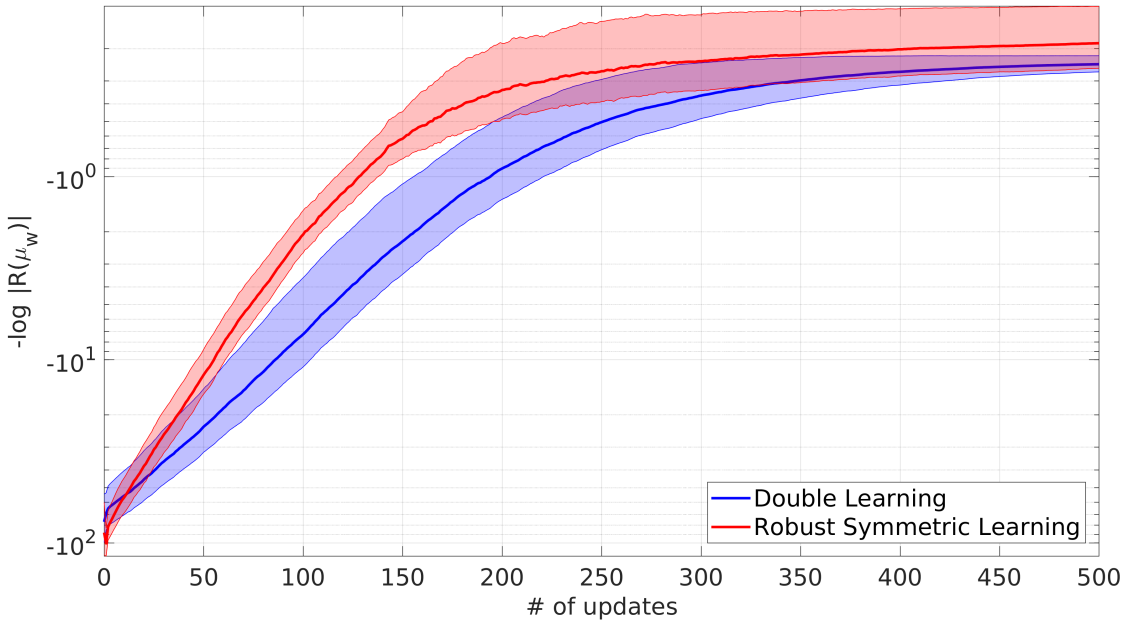


Figure 5.24: $R(\mu_{\boldsymbol{w}})$ *convergence in Test 2, in the case of reference path belonging to an unknown cylinder (values in logarithmic scale).*

Table 5.6: $R(\mu_{\boldsymbol{w}})$ *values in Test 2, for path belonging to a plane, with unknown symmetry's parameters.*

	START	*10 updates*	*50 updates*	*100 updates*	*200 updates*	*500 updates*
DL	-165.3 $\pm$ 32.5	-110.3 $\pm$ 26.9	-63.6 $\pm$ 20.2	-31.2 $\pm$ 13.1	-8.0 $\pm$ 5.3	-1.25 $\pm$ 0.79
RSL	-169.3 $\pm$ 34.4	-82.2 $\pm$ 11.2	-31.2 $\pm$ 4.3	-8.7 $\pm$ 1.4	-2.2 $\pm$ 0.6	-1.04 $\pm$ 0.18

Table 5.7: $R(\mu_{\boldsymbol{w}})$ *values in Test 2, for path belonging to a sphere, with unknown symmetry's parameters.*

	START	*10 updates*	*50 updates*	*100 updates*	*200 updates*	*500 updates*
DL	-61.6 $\pm$ 10.4	-37.4 $\pm$ 8.6	-12.9 $\pm$ 4.0	-2.7 $\pm$ 1.0	-0.37 $\pm$ 0.12	-0.20 $\pm$ 0.01
RSL	-64.9 $\pm$ 15.3	-43.0 $\pm$ 15.1	-7.4 $\pm$ 3.8	-1.1 $\pm$ 0.4	-0.19 $\pm$ 0.03	-0.12 $\pm$ 0.02

Table 5.8: $R(\mu_{\boldsymbol{w}})$ *values in Test 2, for path belonging to a cylinder, with unknown symmetry's parameters.*

	START	*10 updates*	*50 updates*	*100 updates*	*200 updates*	*500 updates*
DL	-76.7 $\pm$ 22.5	-55.4 $\pm$ 15.1	-23.2 $\pm$ 8.8	-7.3 $\pm$ 3.7	-0.9 $\pm$ 0.4	-0.24 $\pm$ 0.02
RSL	-87.9 $\pm$ 27.9	-56.6 $\pm$ 12.5	-12.1 $\pm$ 3.3	-2.1 $\pm$ 0.5	-0.3 $\pm$ 0.1	-0.19 $\pm$ 0.07

Tests ran in simulation seem to confirm that the use of symmetries in the learning of bimanual task can be a way to increase the speed of the update process. Approaches like SL and RSL shows promising results in comparison to the standard approach involving two separate ProMPs. They generally show a faster convergence in the early phases of learning, using half of the amount of parameters necessary to represent two independent movements. The next step is to validate these results, obtained in simulation, in a real case application involving the execution of a more complex bimanual task.

5.4 Experiment: Folding a towel

The symmetric approach to bimanual task learning is now tested with a real-case application. The aim is to make two robotic arms learn how to fold a towel. In order to evaluate the quality of the execution, the experimental setup includes a rooftop-placed *Kinect* camera catching the workspace, providing RGB colors and depth information. Initial ProMPs and symmetry plane estimation have been built from some demonstrations of the task, obtained by kinesthetic teaching to robots how to move to fold the towel on a table (Fig. 5.25). Robotic arms have been equipped with two end-effector grippers allowing to hold firmly the cloth while moving.

Figure 5.25: *One of the initial demonstrations used to kinesthetically teach robots how to fold a towel.*

$M = 15$ basis functions for each direction have been chosen (i.e., $3 \cdot M = 45$ weights in total), with an amplitude parameter of $h = 0.0167$. The regularization term coefficient used in REPS is $\lambda = 10^{-6}$, such that covariance matrix does not need too much updates to converge. The learning process consists of 5 policy updates of 12 rollouts each, with a reuse of the previous 12 sampled weights and rewards in the PS update. Inverse kinematics is used to convert end-effector positions into joint trajectories, which then a computed-torque controller [18] would compliantly track. A main C++ code program is in charge of managing executions of the sequences of rollouts and the storage of data. From the main program, MATLAB functions are called to deal with the sampling and update of ProMPs.

As reward, it is adopted a function that penalizes large joint accelerations, together with an indicator of how well the towel has been folded. In the definition of the reward function it is taken into account how well the resulting shape approximate a rectangle, and how many wrinkles the towel shows. Therefore, the

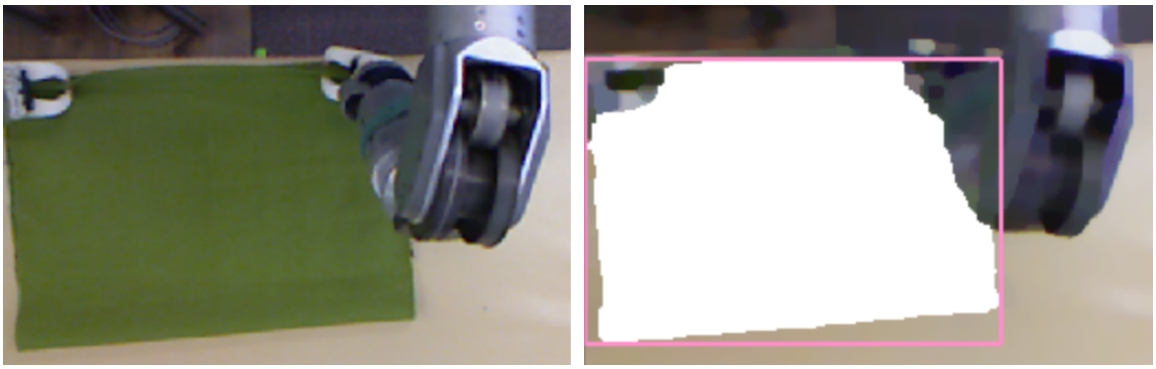
reward function used is:

$$\mathrm{R} = \mathrm{R}_{acc} + \mathrm{R}_{shape} + \mathrm{R}_{wrinkle} \tag{5.9}$$

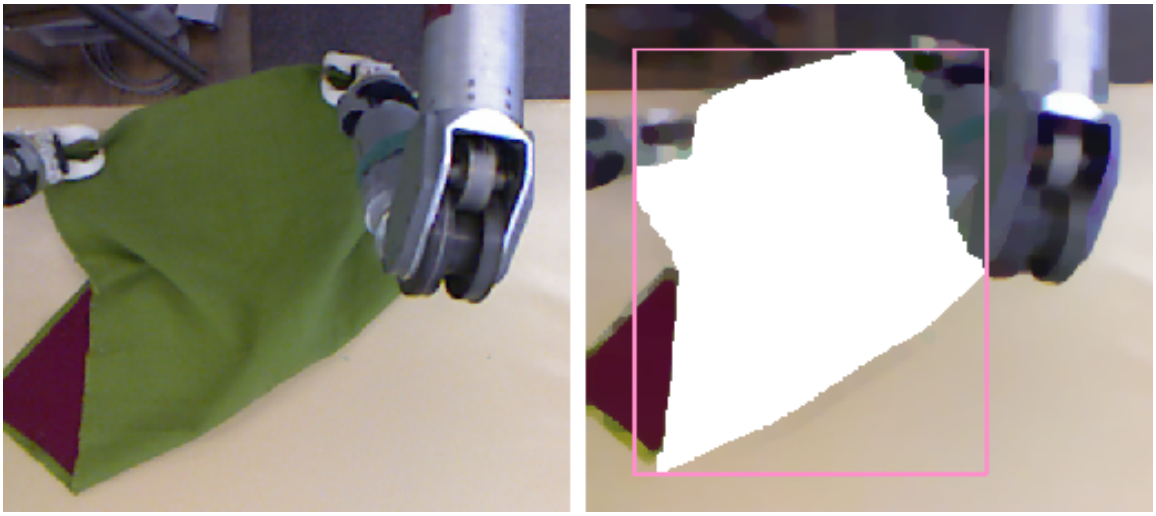
where R_{acc} is a term penalizing large acceleration commands at joint level to avoid sudden movements. The correctness of shape is evaluated by color-segmenting the towel image on the table and fitting a bounding rectangle to the obtained result. It was used a towel with sides of different colors, one in green and one in red. Only green color is considered in the shape-detection in order to have only the side of this color visible from outside after the folding. R_{shape} has a large penalizing value if the result after motion does not have a rectangular figure on the table (Fig. 5.26), and it is expressed as:

$$\mathrm{R}_{shape} = -\frac{\#\text{ rectangle pixels}}{\#\text{towel pixels}} \left[(A - A_{ref})^2 + (B - B_{ref})^2 \right] \tag{5.10}$$

where *A*, *B* are the measured side lengths of the bounding rectangle, and A_{ref} , B_{ref} ref are their reference values, given the towel dimensions. $\mathrm{R}_{wrinkle}$ penalizes the outcome if the towel presents too many wrinkles after the folding (Fig. 5.27) and it is computed using the code available from [19].

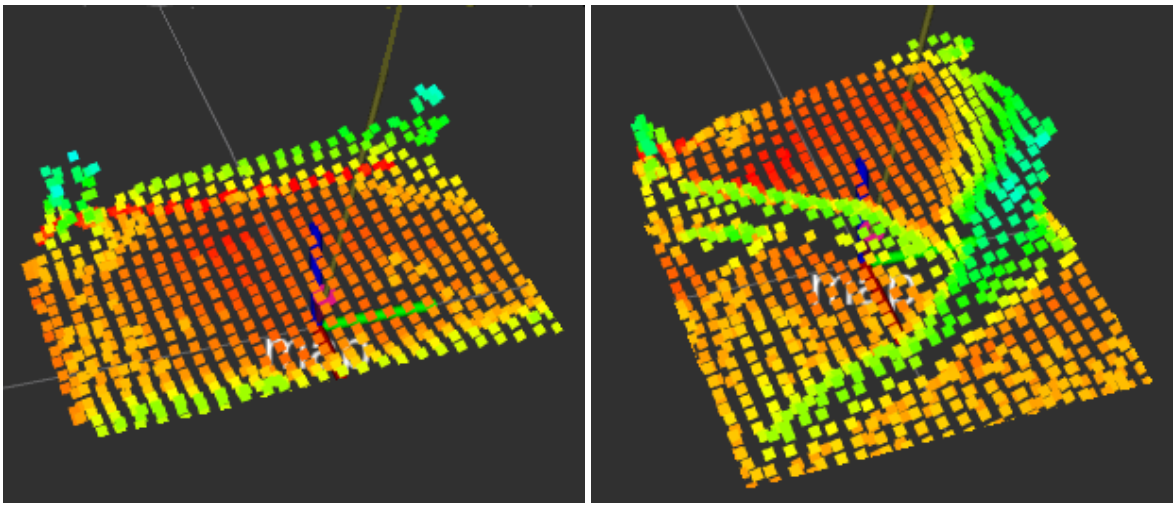


(a) *Camera image and color segmentation of a well-folded towel.*



(b) *Camera image and color segmentation of a bad-folded towel.*

Figure 5.26: *Color segmentation of a towel after two different folding attempts. The green color is segmented and the number of green pixels within the smallest rectangle containing it is counted. The ratio of green pixels w.r. to the total number of pixels within the bounding rectangle is used for the reward function.*



(a) *Depth information from a well-folded towel.* (b) *Depth information from a bad-folded towel.*

Figure 5.27: *Depth image visualization of the towel after two different folding attempts. The mean gradient of the depth was used as a "wrinkleness" indicator.*

RSL and DL approaches have been both tested with this task, succeeding both in folding correctly the towel at the end of the learning process. Exploration of parameter space was problematic in the first iterations of the movement policy due to the physical constraints of task. In fact, movement variability cannot be too big, because if the two arms go too much far one from the other, grippers cannot hold the towel anymore. These situations lead to some bad rollouts from initial policies, in particular in DL, when the two arms move independently one from the other. After some updates this issue is not a problem anymore, thanks to the convergence to more effective folding movements. Evolution of reward values throughout the learning process are shown in Fig. 5.28 and Table 5.9.

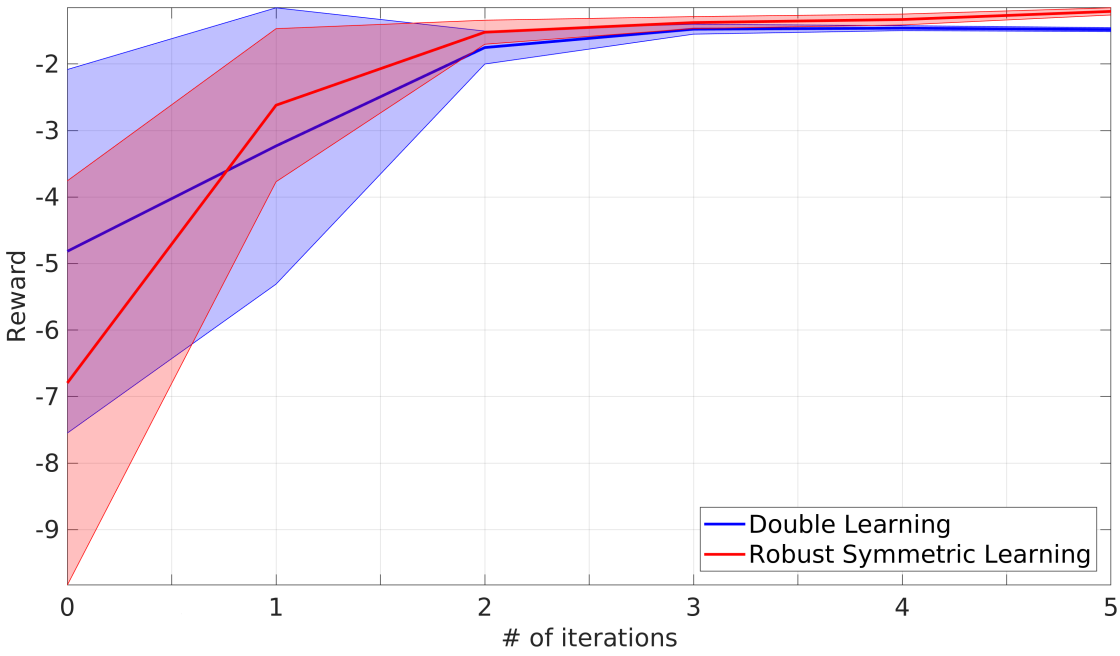


Figure 5.28: *Reward values convergence during the folding experiment.*

Table 5.9: *Reward values obtained during the folding experiment.*

	START	*1 update*	*2 updates*	*3 updates*	*4 updates*	*5 updates*
DL	-4.8 $\pm$ 2.7	-3.2 $\pm$ 2.1	-1.75 $\pm$ 0.24	-1.48 $\pm$ 0.07	-1.46 $\pm$ 0.04	-1.48 $\pm$ 0.03
RSL	-6.8 $\pm$ 3.0	-2.6 $\pm$ 1.2	-1.5 $\pm$ 0.2	-1.37 $\pm$ 0.09	-1.33 $\pm$ 0.08	-1.21 $\pm$ 0.05

RSL at the beginning shows worse results than DL, that is able to model better movements of the second robot arm thanks to its higher amount of parameters. This initial disadvantage is recovered in just one learning update, confirming a faster convergence speed for RSL method, together with a more significant reduction in reward's variance than the one obtained with DL. Furthermore DL converges to a slightly worse reward for the final policy. However the variance in data is very big, and these results should be thoroughly confirmed with further experiments.

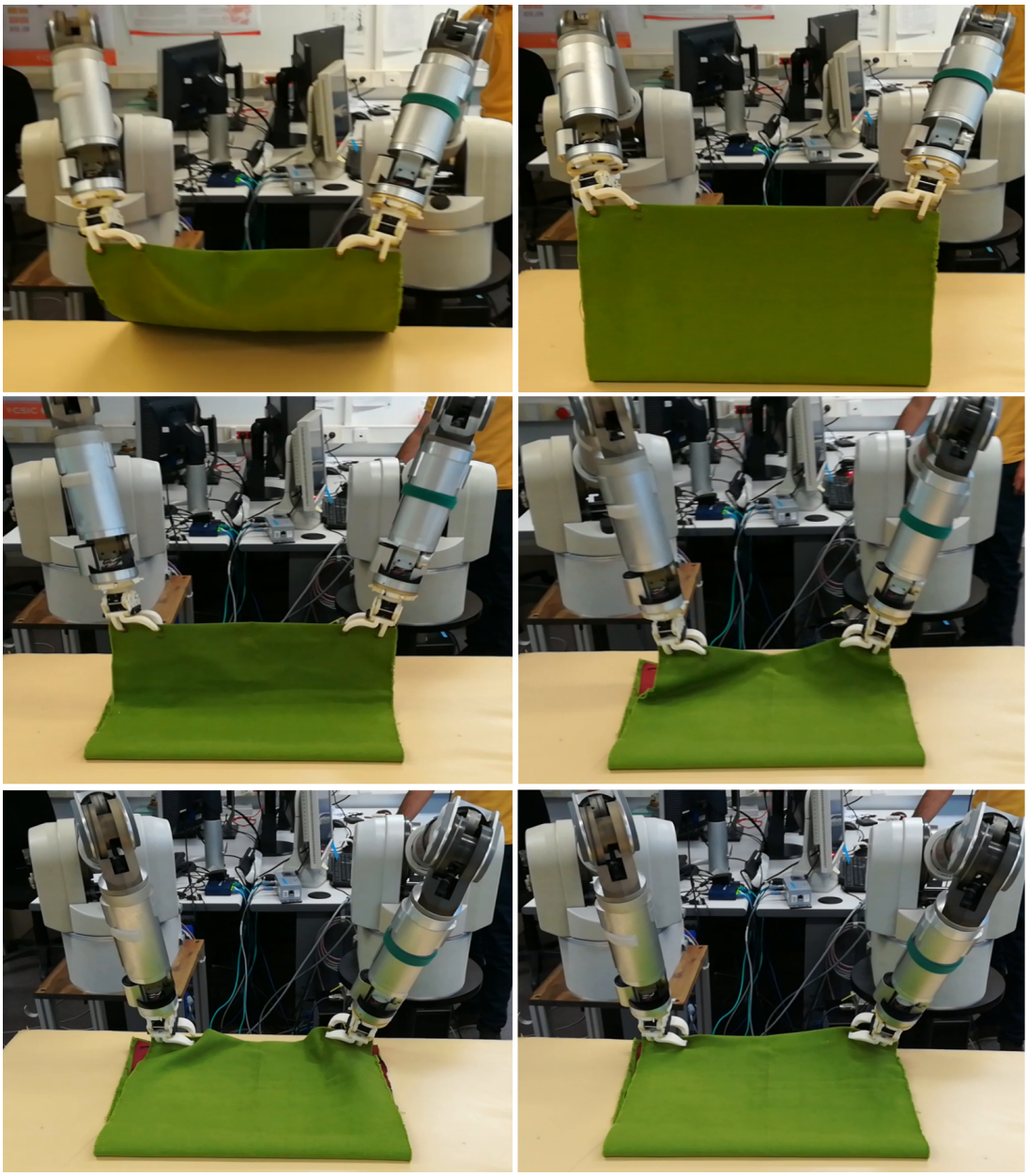


Figure 5.29: *Execution of a rollout from the final policy obtained from RSL.*

Conclusions

The use of symmetries in representation of bimanual robotic movements has proved to be effective in simplifying the dimensionality of policy parameter space, although, standard approach representation (that employs two distinct policies for each robots) could generally fit more finely robotic trajectories, due to its higher number of parameters. The great advantage of the proposed approach is the fastest convergence during learning process. Thanks to the reduced amount of parameters used and the policy exploration phase limited to symmetric trajectories, symmetric learning manages to reach good results with few updates, reducing the time needed and costs. Other significant results of this work are the symmetrization methods derived for multivariate Gaussian distributions, that can see applications also outside of the context of movement representation.

Future works

A first evolution that can be considered is to extend the symmetric approach proposed to other typologies of policy representation, like DMPs, and to other reinforcement learning algorithms, in order to organize a more structured theory on learning symmetric bimanual robotic tasks. Other developments could involve an analysis of symmetries in larger dimensional spaces, such as joint space, analogous to the one developed here for end-effector trajectories in space. Working in joint space, for example, will allow to not rely anymore to a precise characterization of the robot kinematics, and to control more straightforwardly robot arms. An interesting additional work can be the implementation of context variables to the movement representations, in order to make learned symmetric policies more adaptable to different goals and environments.

Bibliography

[1] B. D. Argall, S. Chernova, M. Veloso, B. Browning, "A survey of robot learning from demonstration". *Robotics and Autonomous Systems*, 57(5), pp. 469-483, 2009.

[2] M.P. Deisenroth, G. Neumann and J. Peters, "A Survey on Policy Search for Robotics". *Foundations and Trends in Robotics*, 2(1–2), pp. 1–142, 2011.

[3] S. Schaal, J. Peters, J.Nakanishi, and A. Ijspeert, "Learning Movement Primitives". *International Symposium on Robotics Research*, pp. 561-572, 2003.

[4] M. Spong, S. Hutchinson, M. Vidyasagar, "Robot Modeling and Control". John Wiley and Sons, 2006.

[5] B. Siciliano, L. Sciavicco, L. Villani, G. Oriolo, "Robotics - Modelling, Planning and Control". Springer, 2009.

[6] W.T. Townsend, J.K. Salisbury, "Mechanical Design for Whole-Arm Manipulation". P. Dario, G. Sandini, and P. Aebischer (eds) *Robots and Biological Systems: Towards a New Bionics?* of NATO ASI Series 102, pp. 153–164 Springer, 1993.

[7] Barret Technology, LLC. *Customer Support WAM*: support.barrett.com/wiki/WAM.

[8] A. Colomé, "Smooth Inverse Kinematics Algorithms for Serial Redundant Robots". Institut de Robotica i Informatica Industrial (IRI), Universitat Politecnica de Catalunya (UPC), Barcelona, Spain, 2011.

[9] M. Shimizu, H. Kakuya, W.K. Yoon, K. Kitagaki, and K. Kosuge, "Analytical inverse kinematic computation for 7-dof redundant manipulators with joint limits and its application to redundancy resolution". IEEE *Transactions on Robotics*, 24(5), pp. 1131–1142, 2008.

[10] G.K. Singh, and J. Claassens, "An analytical solution for the inverse kinematics of a redundant 7dof manipulator with link offsets". *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 2976–2982, 2010.

[11] P. Corke, "Robotics, Vision and Control Fundamental Algorithms In MATLAB: Second Edition". Springer, 2017.

[12] S. Schaal, "Dynamic Movement Primitives - A Framework for Motor Control in Humans and Humanoid Robotics". H. Kimura, K. Tsuchiya, A. Ishiguro, H. Witte (eds) *Adaptive Motion of Animals and Machines*, Springer, 2006.

[13] A. Paraschos, C. Daniel, J. Peters, and G. Neumann, "Probabilistic Movement Primitives". *Advances in Neural Information Processing Systems (NIPS)*, 2013.

[14] A. S. Householder, "Unitary Triangularization of a Nonsymmetric Matrix". *Journal of the ACM*, 5(4), pp. 339–342, 1958.

[15] S. Kullback, "Information Theory and Statistics". John Wiley and Sons, 1959.

[16] C.M. Bishop, "Pattern Recognition and Machine Learning". Springer, 2006.

[17] J. Peters, K. Mülling, and Y. Altün, "Relative Entropy Policy Search". *24th National Conference on Artificial Intelligence*, 15, pp. 182-189, 2011.

[18] A. Colomé, A. Planells and C. Torras, "A friction-model-based framework for reinforcement learning of robotic tasks in non-rigid environments". *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 5649-5654, 2015.

[19] A. Ramisa, G. Alenyà, F. Moreno-Noguer, and C. Torras. "A 3D descriptor to detect task-oriented grasping points in clothing". *Pattern Recognition*, 60, pp. 936-948, 2016.

Derivation of the Dual Function

Given the optimization problem (5.2), the associated Lagrangian can be computed as:

$$\mathcal{L} = \int \pi(\boldsymbol{w})\mathrm{R}(\boldsymbol{w})d\boldsymbol{w} + \eta\left(\epsilon - \int \pi(\boldsymbol{w})\log\frac{\pi(\boldsymbol{w})}{q(\boldsymbol{w})}d\boldsymbol{w}\right) + \lambda\left(1 - \int \pi(\boldsymbol{w})d\boldsymbol{w}\right) \tag{A.1}$$

Differentiating with respect to $\pi(\boldsymbol{w})$ (and omitting $\boldsymbol{w}$ for simplicity) it is obtained

$$\frac{\partial\mathcal{L}}{\partial\pi} = \mathrm{R} - \eta(\log\pi - \log q + 1) - \lambda \tag{A.2}$$

which, setting $\frac{\partial\mathcal{L}}{\partial\pi} = 0$ and isolating $\log\pi$, becomes

$$\log\pi = \frac{R}{\eta} + \log q - \frac{\eta + \lambda}{\eta} \tag{A.3}$$

and setting $Z = e^{\frac{\eta+\lambda}{\eta}}$,

$$\pi = Z^{-1}\, q\, e^{\mathrm{R}/\eta} \tag{A.4}$$

from which, integrating for $\boldsymbol{w}$ and given that $1 = \int \pi d\boldsymbol{w}$, it derives

$$Z = \int q\, e^{\mathrm{R}/\eta} d\boldsymbol{w}. \tag{A.5}$$

Inserting now (A.4) into (A.2) an expression for the dual function is obtained.

$$g(\eta,\lambda) = \int Z^{-1}\, q\, \mathrm{R}\, e^{\mathrm{R}/\eta} d\boldsymbol{w} + \eta\epsilon - \eta\int Z^{-1}\, q\, e^{\mathrm{R}/\eta}\left(\log Z^{-1} + \mathrm{R}/\eta\right) d\boldsymbol{w} + \lambda\left(1 - \int Z^{-1}\, q\, e^{\mathrm{R}/\eta} d\boldsymbol{w}\right) \tag{A.6}$$

$$= \cancel{\int Z^{-1}\, q\, \mathrm{R}\, e^{\mathrm{R}/\eta} d\boldsymbol{w}} + \eta\epsilon - \eta\int Z^{-1}\, q\, e^{\mathrm{R}/\eta}\log Z^{-1}\, d\boldsymbol{w} - \cancel{\int Z^{-1}\, q\, e^{\mathrm{R}/\eta}\, \mathrm{R}\, d\boldsymbol{w}} + \lambda\left(1 - \int Z^{-1}\, q\, e^{\mathrm{R}/\eta} d\boldsymbol{w}\right) \tag{A.7}$$

$$= \eta\epsilon + \eta\int Z^{-1}\, q\, e^{\mathrm{R}/\eta}\log Z\, d\boldsymbol{w} + \lambda - \lambda\int Z^{-1}\, q\, e^{\mathrm{R}/\eta} d\boldsymbol{w} \tag{A.8}$$

recalling the expression of Z, $\log Z = \left(\frac{\eta+\lambda}{\eta}\right)$, thus

$$= \eta\epsilon + \eta\int Z^{-1}\, q\, e^{\mathrm{R}/\eta} d\boldsymbol{w} + \cancel{\lambda\int Z^{-1}\, q\, e^{\mathrm{R}/\eta} d\boldsymbol{w}} + \lambda - \cancel{\lambda\int Z^{-1}\, q\, e^{\mathrm{R}/\eta} d\boldsymbol{w}} \tag{A.9}$$

and being Z^{-1} a constant, it can be moved out of the integral

$$= \eta\epsilon + \eta\, Z^{-1}\int q\, e^{\mathrm{R}/\eta} d\boldsymbol{w} + \lambda. \tag{A.10}$$

The integral expression is actually equal to Z from (A.5), so

$$g(\eta,\lambda) = \eta\epsilon + \eta + \lambda. \tag{A.11}$$

Matching the two expressions of Z it is possible to isolate λ and obtain

$$\lambda = \eta \log \int q \, e^{\mathrm{R}/\eta} \, d\boldsymbol{w} - \eta \tag{A.12}$$

. Finally, inserting (A.12) into (A.11) the dual function is obtained:

$$g(\eta) = \eta\epsilon + \eta \log \int q(\boldsymbol{w}) e^{\frac{\mathrm{R}(\boldsymbol{w})}{\eta}} \, d\boldsymbol{w} \tag{A.13}$$

The integral can be replaces over a sum of K sample:

$$g(\eta) = \eta\epsilon + \eta \log \left[\frac{1}{K} \sum_{k=1}^{K} e^{\frac{\mathrm{R}(\boldsymbol{w}_k)}{\eta}} \right] \tag{A.14}$$

B.1 WAM arm kinematic model

A kinematic model of WAM arm can be obtained in MATLAB (with Robotics Toolbox), from the D-H parameters describing its links, running the following scripts.

```
clear L
% Define joints' D-H parameters
L{1} = link([-pi/2 0 0 0 0],'standard');
L{2} = link([pi/2 0 0 0 0],'standard');
L{3} = link([-pi/2 0.045 0 0.55 0],'standard');
L{4} = link([pi/2 -0.045 0 0 0],'standard');
L{5} = link([-pi/2 0 0 0.3 0],'standard');
L{6} = link([pi/2 0 0 0 0],'standard');
L{7} = link([0 0 0 0.06 0],'standard');
% Create robot model
wam = robot(L, 'WAM Arm', 'Barrett', 'dh params');
clear L
```

After that, robot model `wam` can be used with functions `fkine` and `ikine` to compute forward and inverse kinematics respectively.

B.2 Symmetrization of Gaussian distributions

`symm_3Dgauss` computes the symmetric of a 3D Gaussian distribution with respect to a given plane.

```
function [mu_s,Sigma_s] = symm_3Dgauss(mu,Sigma,a,b,c,d)
% symm_3Dgauss Planar symmetric of a 3D Gaussian.
%
%   [mu_s,Sigma_s] = symm_3Dgauss(mu,Sigma,a,b,c,d) computes the symmetric
%   of a 3D Gaussian distribution (mu,Sigma) with respect to the plane
%   defined by the parameters a,b,c,d (ax+by+cz+d=0). The resulting
%   symmetric Gaussian distribution is given by (mu_s,Sigma_s).

% Get covariance matrix eigenvectors and eigenvalues
[V,D] = eig(Sigma);

% Eigenvectors extreme points
ev1P = mu + D(1,1)*V(:,1);
ev2P = mu + D(2,2)*V(:,2);
```

```
ev3P = mu + D(3,3)*V(:,3);

% The parametrized expression of the line perpendicular to the plane going
% through P is: P + t*<a,b,c> (t is the varying parameter)

% Symmetric of the center point mu
tc = -([a b c]*mu+d)/(a^2+b^2+c^2);          % projection: mu+tc*<a,b,c>
mu_s = mu +2*tc*[a b c]';                    % symmetric of the mean mu

% Symmetric of eigenvectors extreme points
tev1 = -([a b c]*ev1P+d)/(a^2+b^2+c^2);      % projection: ev1P+tev1*<a,b,c>
ev1P_s = ev1P +2*tev1*[a b c]';              % symmetric of ev1P
tev2 = -([a b c]*ev2P+d)/(a^2+b^2+c^2);      % projection: ev2P+t3v2*<a,b,c>
ev2P_s = ev2P +2*tev2*[a b c]';              % symmetric of ev2P
tev3 = -([a b c]*ev3P+d)/(a^2+b^2+c^2);      % projection: ev3P+tev3*<a,b,c>
ev3P_s = ev3P +2*tev3*[a b c]';              % symmetric of ev3P

% Calculate symmetric eigenvectors
V_s = [(ev1P_s-mu_s)./D(1,1) (ev2P_s-mu_s)./D(2,2) (ev3P_s-mu_s)./D(3,3)];
Sigma_s = V_s*D*V_s'; % symmetric of the covariance matrix Sigma
end
```

`sphericSymm_3Dgauss` computes the symmetric of a 3D Gaussian distribution with respect to a given sphere.

```
function [mu_s,Sigma_s] = sphericSymm_3Dgauss(mu,Sigma,param)
% sphericSymm_3Dgauss Spherical symmetric of a 3D Gaussian.
%
%   [mu_s,Sigma_s] = sphericSymm_3Dgauss(mu,Sigma,param) computes the
%   symmetric of a 3D Gaussian distribution (mu,Sigma) with respect to the
%   sphere centered in (xc,yc,zc) and radius r (param=[xc;yc;zc;r]).
%   The resulting symmetric Gaussian distribution is (mu_s,Sigma_s).

% Find the plane tangent to the surface through the mu projection
center = param(1:3);
r = param(4);

vc = mu-center;                   % vector of radius going through mu
tc = r/sqrt((sum(vc.^2)));        % parameter associated to the projection
mu_pr = center +tc*vc;            % projection of the point in the sphere
dist = sqrt(sum((mu_pr-mu).^ 2)); % distance from the surface

a=vc(1); b=vc(2); c=vc(3); d=-vc'*mu_pr;

% Initial planar symmetrization
[mu_s,Sigma_s] = symm_3Dgauss(mu,Sigma,a,b,c,d);

[V,D] = eig(Sigma_s);
v1 = V(:,1);
v2 = V(:,2);
v3 = V(:,3);
```

```
% Vectors projection
temp = null(vc');
r1 = temp(:,1); r2 = temp(:,2);    % tangent vectors to the sphere

pr11 = dot(v1,r1)/dot(r1,r1)*r1;   % v1 projection onto r1
pr12 = dot(v1,r2)/dot(r2,r2)*r2;   % v1 projection onto r2
nn1 = v1 - pr11 - pr12;            % pr11 + pr12 + nn1 = v1

pr21 = dot(v2,r1)/dot(r1,r1)*r1;   % v2 projection onto r1
pr22 = dot(v2,r2)/dot(r2,r2)*r2;   % v2 projection onto r2
nn2 = v2 - pr21 - pr22;            % pr21 + pr22 + nn2 = v2

pr31 = dot(v3,r1)/dot(r1,r1)*r1;   % v3 projection onto r1
pr32 = dot(v3,r2)/dot(r2,r2)*r2;   % v3 projection onto r2
nn3 = v3 - pr31 - pr32;            % pr31 + pr32 + nn3 = v3

% Curvature scaling factor
if(dist<r)
    k = (r+dist)/(r-dist);         % mu inside the sphere
else
    k = (r-dist)/(r+dist);         % mu outside the sphere
end

% New eigenvectors
pr11 = k*pr11; pr12 = k*pr12; v1_new = pr11 + pr12 + nn1;
pr21 = k*pr21; pr22 = k*pr22; v2_new = pr21 + pr22 + nn2;
pr31 = k*pr31; pr32 = k*pr32; v3_new = pr31 + pr32 + nn3;

k1 = norm(v1_new)/norm(v1);
k2 = norm(v2_new)/norm(v2);
k3 = norm(v3_new)/norm(v3);

% Eigenvectors normalization
v1_new = v1_new/norm(v1_new);
v2_new = v2_new/norm(v2_new);
v3_new = v3_new/norm(v3_new);

V = [v1_new v2_new v3_new];

% New eigenvalues
D(1,1) = D(1,1)*k1;
D(2,2) = D(2,2)*k2;
D(3,3) = D(3,3)*k3;

Sigma_s = V*D*V^(-1);
Sigma_s = (Sigma_s+Sigma_s')/2;
end
```

`cylindricSymm_3Dgauss` computes the symmetric of a 3D Gaussian distribution with respect to a given cylinder.

```
function [mu_s,Sigma_s] = cylindricSymm_3Dgauss(mu,Sigma,param)
% cylindricSymm_3Dgauss Cylindrical symmetric of a 3D Gaussian.
%
%   [mu_s,Sigma_s] = symm_3Dgauss(mu,Sigma,param) computes the symmetric
%   of a 3D Gaussian distribution (mu,Sigma) with respect to the vertical
%   cylinder centered in (xc,yc) with radius r (param=[xc;yx;r]).
%   The resulting symmetric Gaussian distribution is (mu_s,Sigma_s).

% Find the plane tangent to the surface through the mu projection
center = [param(1:2); mu(3)];
r = param(3);

vc = mu-center;                    % vector of radius going through mu
tc = r/sqrt((sum(vc.^2)));         % parameter associated to the projection
mu_pr = center +tc*vc;             % projection of the point in the sphere
dist = sqrt(sum((mu_pr-mu).^ 2)); % distance from the surface

a=vc(1); b=vc(2); c=vc(3); d=-vc'*mu_pr;

% Initial planar symmetrization
[mu_s,Sigma_s] = symm_3Dgauss(mu,Sigma,a,b,c,d);

[V,D] = eig(Sigma_s);
v1 = V(:,1);
v2 = V(:,2);
v3 = V(:,3);

% Vectors projection
vp = [vc(2); -vc(1); vc(3)];       % tangent vector to the cylinder

pr1 = dot(v1,vp)/dot(vp,vp)*vp;    % v1 projection onto vp
nn1 = v1 - pr1;                    % pr1 + nn1 = v1

pr2 = dot(v2,vp)/dot(vp,vp)*vp;    % v2 projection onto vp
nn2 = v2 - pr2;                    % pr2 + nn2 = v2

pr3 = dot(v3,vp)/dot(vp,vp)*vp;    % v3 projection onto vp
nn3 = v3 - pr3;                    % pr3 + nn3 = v3

% Curvature scaling factor
if(dist<r)
    k = (r+dist)/(r-dist);         % mu inside the sphere
else
    k = (r-dist)/(r+dist);         % mu outside the sphere
end

% New eigenvectors
pr1 = k*pr1; v1_new = pr1 + nn1;
pr2 = k*pr2; v2_new = pr2 + nn2;
pr3 = k*pr3; v3_new = pr3 + nn3;

k1 = norm(v1_new)/norm(v1);
```

```
k2 = norm(v2_new)/norm(v2);
k3 = norm(v3_new)/norm(v3);

% Eigenvectors normalization
v1_new = v1_new/norm(v1_new);
v2_new = v2_new/norm(v2_new);
v3_new = v3_new/norm(v3_new);

V = [v1_new v2_new v3_new];

% New eigenvalues
D(1,1) = D(1,1)*k1;
D(2,2) = D(2,2)*k2;
D(3,3) = D(3,3)*k3;

Sigma_s = V*D*V^(-1);
Sigma_s = (Sigma_s+Sigma_s')/2;
end
```

B.3 Symmetrization of ProMPs

`symm_ProMP` computes the symmetric of a ProMP with respect to a given plane.

```
function [mu_w_symm,S_w_symm] = symm_ProMP(mu_w,S_w,basis_f,plane,T)
% symm_ProMP Planar symmetric of a ProMP defined for a 3D movement.
%
%   [mu_w_symm,S_w_symm] = symm_ProMP(mu_w,S_w,basis_f,plane,center,T)
%   gets symmetric of the given ProMP: mu_w,S_w (in the reference
%   frame of the second robot-arm) with respect to the plane in 3D
%   space defined by the parameters a,b,c,d inside 'plane'.
%   The resulting symmetric ProMP is given by (mu_w_symm,S_w_symm).
%   [This function uses symm_3Dgauss]

a = plane(1); b = plane(2); c = plane(3); d = plane(4);

Nt = length(basis_f);
Psi = blkdiag(basis_f,basis_f,basis_f);

mean_tr = Psi'*mu_w;
mean_xyz = [mean_tr(1:Nt)';mean_tr(Nt+1:2*Nt)';mean_tr(2*Nt+1:3*Nt)'];

A = []; B = [];
S_w_symm_m1 = zeros(3*size(basis_f,1));
for t=1:Nt
    Psi_t(:,:,t) = blkdiag(basis_f(:,t),basis_f(:,t),basis_f(:,t));

    % Covariance of XYZ trajectory at time t
    S_xyz(:,:,t) = Psi_t(:,:,t)'*S_w*Psi_t(:,:,t);

    % Symmetrize
```

```
    [mean_xyz_symm_1(:,t),S_xyz_symm_1(:,:,t)] = ...
        symm_3Dgauss(mean_xyz(:,t),S_xyz(:,:,t),a,b,c,d);

    % Change of frame
    aux = T\[mean_xyz_symm_1(:,t); 1];
    mean_xyz_symm(:,t) = aux(1:3);
    R = T(1:3,1:3);
    S_xyz_symm(:,:,t) = R\S_xyz_symm_1(:,:,t);

    % Block matrices to compute S_w_symm
    A = [A; S_xyz_symm(:,:,t)*pinv(Psi_t(:,:,t))];
    B = [B; Psi_t(:,:,t)'];

end
mean_tr_symm = [mean_xyz_symm(1,:)'; ...
                mean_xyz_symm(2,:)'; ...
                mean_xyz_symm(3,:)'];

mu_w_symm = pinv(Psi')*mean_tr_symm;
S_w_symm = pinv(B)*A;

% Force symmetry
S_w_symm = (S_w_symm+S_w_symm')/2;

mu_w_symm = real(mu_w_symm);
S_w_symm = real(S_w_symm);
end
```

`sphericSymm_ProMP` computes the symmetric of a ProMP with respect to a given sphere.

```
function [mu_w_symm,S_w_symm] = sphericSymm_ProMP(mu_w,S_w,basis_f,param,T)
% sphericSymm_ProMP Spheric symmetric of a ProMP defined for a 3D movement.
%
%   [mu_w_symm,S_w_symm] = sphericSymm_ProMP(mu_w,S_w,basis_f,param,T)
%   gets symmetric of the given ProMP: mu_w,S_w (in the reference
%   frame of the second robot-arm) with respect to the sphere
%   defined by the parameters (xc,yc,zc,r) inside 'param'.
%   The resulting symmetric ProMP is given by (mu_w_symm,S_w_symm).
%   [This function uses sphericSymm_3Dgauss]

Nt = length(basis_f);
Psi = blkdiag(basis_f,basis_f,basis_f);

mean_tr = Psi'*mu_w;
mean_xyz = [mean_tr(1:Nt)';mean_tr(Nt+1:2*Nt)';mean_tr(2*Nt+1:3*Nt)'];

A = []; B = [];
S_w_symm_m1 = zeros(3*size(basis_f,1));
for t=1:Nt
    Psi_t(:,:,t) = blkdiag(basis_f(:,t),basis_f(:,t),basis_f(:,t));

    % Covariance of XYZ trajectory at time t
```

```
    S_xyz(:,:,t) = Psi_t(:,:,t)'*S_w*Psi_t(:,:,t);

    % Symmetrize
    [mean_xyz_symm_1(:,t),S_xyz_symm_1(:,:,t)] = ...
        sphericSymm_3Dgauss(mean_xyz(:,t),S_xyz(:,:,t),param);

    % Change of frame
    aux = T\[mean_xyz_symm_1(:,t); 1];
    mean_xyz_symm(:,t) = aux(1:3);
    R = T(1:3,1:3);
    S_xyz_symm(:,:,t) = R\S_xyz_symm_1(:,:,t);

    % Block matrices to compute S_w_symm
    A = [A; S_xyz_symm(:,:,t)*pinv(Psi_t(:,:,t))];
    B = [B; Psi_t(:,:,t)'];

end
mean_tr_symm = [mean_xyz_symm(1,:)'; ...
                mean_xyz_symm(2,:)'; ...
                mean_xyz_symm(3,:)'];

mu_w_symm = pinv(Psi')*mean_tr_symm;
S_w_symm = pinv(B)*A;

% Force symmetry
S_w_symm = (S_w_symm+S_w_symm')/2;

mu_w_symm = real(mu_w_symm);
S_w_symm = real(S_w_symm);
end
```

`cylindricSymm_ProMP` computes the symmetric of a ProMP with respect to a given cylinder.

```
function [mu_w_symm,S_w_symm] = cylindricSymm_ProMP(mu_w,S_w,basis_f,param,T)
% cylindricSymm_ProMP Cylindric symmetric of a ProMP defined for a 3D movement.
%
%   [mu_w_symm,S_w_symm]=cylindricSymm_ProMP(mu_w,S_w,basis_f,param,T)
%   gets symmetric of the given ProMP: mu_w,S_w (in the reference
%   frame of the second robot-arm) with respect to the vertical
%   cylinder defined by the parameters (xc,yc,r) inside 'param'.
%   The resulting symmetric ProMP is given by (mu_w_symm,S_w_symm).
%   [This function uses cylindricSymm_3Dgauss]

Nt = length(basis_f);
Psi = blkdiag(basis_f,basis_f,basis_f);

mean_tr = Psi'*mu_w;
mean_xyz = [mean_tr(1:Nt)';mean_tr(Nt+1:2*Nt)';mean_tr(2*Nt+1:3*Nt)'];

A = []; B = [];
S_w_symm_m1 = zeros(3*size(basis_f,1));
for t=1:Nt
```

```
    Psi_t(:,:,t) = blkdiag(basis_f(:,t),basis_f(:,t),basis_f(:,t));

    % Covariance of XYZ trajectory at time t
    S_xyz(:,:,t) = Psi_t(:,:,t)'*S_w*Psi_t(:,:,t);

    % Symmetrize
    [mean_xyz_symm_1(:,t),S_xyz_symm_1(:,:,t)] = ...
        cylindricSymm_3Dgauss(mean_xyz(:,t),S_xyz(:,:,t),param);

    % Change of frame
    aux = T\[mean_xyz_symm_1(:,t); 1];
    mean_xyz_symm(:,t) = aux(1:3);
    R = T(1:3,1:3);
    S_xyz_symm(:,:,t) = R\S_xyz_symm_1(:,:,t);

    % Block matrices to compute S_w_symm
    A = [A; S_xyz_symm(:,:,t)*pinv(Psi_t(:,:,t))];
    B = [B; Psi_t(:,:,t)'];

end
mean_tr_symm = [mean_xyz_symm(1,:)'; ...
                mean_xyz_symm(2,:)'; ...
                mean_xyz_symm(3,:)'];

mu_w_symm = pinv(Psi')*mean_tr_symm;
S_w_symm = pinv(B)*A;

% Force symmetry
S_w_symm = (S_w_symm+S_w_symm')/2;

mu_w_symm = real(mu_w_symm);
S_w_symm = real(S_w_symm);
end
```

B.4 Estimation of symmetry surfaces

B.4.1 Plane estimation

Estimation of symmetry plane parameters *(a,b,c,d)* can be performed using PCA, as it is done in the following script (where `TRAJ1` and `TRAJ2_1` are the two sets containing demonstrated trajectories both expressed in the reference frame of the first arm, `meanTraj1` and `meanTraj2` are their averages).

```
% Estimation of plane's parameters
midpoints = (meanTraj1(:,1:end)+meanTraj2(:,1:end))./2;
x_mid = midpoints(:,1)'; y_mid = midpoints(:,2)'; z_mid = midpoints(:,3)';

% Principal Component Analysis
[COEFF,~,~] = pca(midpoints); % COEFF: principal component coefficients (by column)
```

```
center = mean(midpoints);      % Center of the plane
v = COEFF(:,3);                % Vector orthogonal to the plane
% First estimation of the symmetry plane: ax+by+cz+d=0
a = v(1); b = v(2); c = v(3); d = -[a b c]*center';

% Parameters covariance
for i=1:Kd
    x1 = TRAJ1{i}(:,1); y1 = TRAJ1{i}(:,2); z1 = TRAJ1{i}(:,3);
    x2 = TRAJ2_1{i}(:,1); y2 = TRAJ2_1{i}(:,2); z2 = TRAJ2_1{i}(:,3);

    mid{i} = [(x1+x2)/2 (y1+y2)/2 (z1+z2)/2];

    [COEFF, ~, LATENT] = pca(mid{i});

    center = mean(mid{i});     % Center of the plane
    v = COEFF(:,3);            % Vector orthogonal to the plane

    % First estimation of the symmetry plane: ax+by+cz+d=0
    a = v(1); b = v(2); c = v(3); d = -[a b c]*center';
    plane{i} = [a;b;c;d];
end

S_plane = 0;
for i=1:Kd
    S_plane = S_plane + (plane{i}-plane_init)*(plane{i}-plane_init)'/(Kd-1);
end
```

B.4.2 Sphere estimation

Estimation of symmetry sphere parameters (x_C, y_C, z_C, R) can be done through optimization techniques (using MATLAB function `fminunc`), as it is done in the following script (where `TRAJ1` and `TRAJ2_1` are the two sets containing demonstrated trajectories both expressed in the reference frame of the first arm, `meanTraj1` and `meanTraj2` are their averages).

```
% Estimation of sphere's parameters
opt = optimoptions(@fminunc,'Display','iter','Algorithm','quasi-newton');
fun = @(X)aggrDist(TRAJ1,TRAJ2_1,X,Kd,Nt);
C_init = fminunc(fun,zeros(3,1),opt);

mean_midpoints = (meanTraj1+meanTraj2)/2;
C_mid = mean(mean_midpoints,1)';

R_init = sqrt((C_new-C_mid)'*(C_new-C_mid));

param_init = [C_init; R_init];

% Parameters covariance
for i=1:Kd
    opt = optimoptions(@fminunc,'Display','iter','Algorithm','quasi-newton');
    fun = @(X)aggrDist_single(TRAJ1{i},TRAJ2_1{i},X,Nt);
```

```
    C1(:,i) = fminunc(fun,zeros(3,1),opt);

    mid = (TRAJ1{i}+TRAJ2_1{i})/2;

    R1(i) = sqrt((C1(:,i)-mean(mid,1)')'*(C1(:,i)-mean(mid,1)'));
end
param1 = [C1; R1];

S_param = 0;
for i=1:Kd
    S_param = S_param + (param1(:,i)-param_init)*(param1(:,i)-param_init)'/(Kd-1);
end
```

`aggrDist` and `aggrDist_single` are used to compute the quadratic distances of a point from the lines defined by end-effectors' positions. They are the functions to be optimized in order to find the estimation of center and radius of the symmetry sphere and their covariance matrix.

```
function D = aggrDist(TRAJ1,TRAJ2_1,Ce,Kd,Nt)
% aggrDist computes the sum of quadratic distances of Ce from the lines
% defined by the two end-effector positions in all the Kd demonstrations
% given in sets TRAJ1 and TRAJ2_1 at each of the Nt time instant.

D = 0;
for i=1:Kd
    x1 = TRAJ1{i}(:,1); y1 = TRAJ1{i}(:,2); z1 = TRAJ1{i}(:,3);
    x2 = TRAJ2_1{i}(:,1); y2 = TRAJ2_1{i}(:,2); z2 = TRAJ2_1{i}(:,3);
    for k=1:Nt
        vk=[x2(k)-x1(k); y2(k)-y1(k); z2(k)-z1(k)]; % direction vector
        a=vk(1); b=vk(2); c=vk(3); % defining also the orthogonal plane
        d=-(a*Ce(1)+b*Ce(2)+c*Ce(3)); % passing through C
        % parametric line: r_k = P1_k + t*vk | tt defines the projection
        tt = -(a*x1(k)+b*y1(k)+c*z1(k)+d)/(a*vk(1)+b*vk(2)+c*vk(3));
        CP = [x1(k);y1(k);z1(k)]+tt*vk;

        dist = (Ce-CP)'*(Ce-CP);
        D = D + dist;
    end
end
end
```

```
function D = aggrDist_single(traj1,traj2_1,Ce,Nt)
% aggrDist_single computes the sum of quadratic distances of Ce from
% the lines defined by the two end-effector positions in demonstrations
% traj1 and traj2 at each of the Nt time instant.

D = 0;

x1 = traj1(:,1); y1 = traj1(:,2); z1 = traj1(:,3);
x2 = traj2_1(:,1); y2 = traj2_1(:,2); z2 = traj2_1(:,3);
```

```
for k=1:Nt
    vk=[x2(k)-x1(k); y2(k)-y1(k); z2(k)-z1(k)]; % direction vector
    a=vk(1); b=vk(2); c=vk(3); % defining also the orthogonal plane
    d=-(a*Ce(1)+b*Ce(2)+c*Ce(3)); % passing through C
    % parametric line: r_k = P1_k + t*vk | tt defines the projection
    tt = -(a*x1(k)+b*y1(k)+c*z1(k)+d)/(a*vk(1)+b*vk(2)+c*vk(3));
    CP = [x1(k);y1(k);z1(k)]+tt*vk;

    dist = (Ce-CP)'*(Ce-CP);
    D = D + dist;
end
end
```

B.4.3 Cylinder estimation

Estimation of symmetry sphere parameters (x_C, y_C, R) can be done through optimization techniques (using MATLAB function `fminunc`), as it is done in the following script (where `TRAJ1` and `TRAJ2_1` are the two sets containing demonstrated trajectories both expressed in the reference frame of the first arm, `meanTraj1` and `meanTraj2` are their averages).

```
% Estimation of cylinder's parameters
opt = optimoptions(@fminunc,'Display','iter','Algorithm','quasi-newton');
fun = @(X)aggrDist2(TRAJ1,TRAJ2_1,X,Kd,Nt);
C_init = fminunc(fun,zeros(2,1),opt);

mean_midpoints = (meanTraj1+meanTraj2)/2;
C_mid = mean(mean_midpoints,1)';

R_init = sqrt((C_init-C_mid(1:2))'*(C_init-C_mid(1:2)));

param_init = [C_init(1:2); R_init];

% Parameters covariance
for i=1:Kd
    opt = optimoptions(@fminunc,'Display','iter','Algorithm','quasi-newton');
    fun = @(X)aggrDist2_single(TRAJ1{i},TRAJ2_1{i},X,Nt);
    C1(:,i) = fminunc(fun,zeros(2,1),opt);

    mid = (TRAJ1{i}+TRAJ2_1{i})/2;

    R1(i) = sqrt((C1(:,i)-mean(mid(:,1:2),1)')'*(C1(:,i)-mean(mid(:,1:2),1)'));
end
param1 = [C1(1:2,:); R1];

S_param = 0;
for i=1:Kd
    S_param = S_param + (param1(:,i)-param_init)*(param1(:,i)-param_init)'/(Kd-1);
end
```

`aggrDist2` and `aggrDist2_single` are used to compute the quadratic distances of a point from the lines defined by end-effectors' positions in the XY plane. They are the functions to be optimized to find the estimation of center and radius of the symmetry cylinder and their covariance matrix.

```
function D = aggrDist2(TRAJ1,TRAJ2_1,Ce,Kd,Nt)
% aggrDist2 computes the sum of quadratic distances of Ce from the lines
% defined by the two end-effector positions in all the Kd demonstrations
% given in sets TRAJ1 and TRAJ2_1 at each of the Nt time instant, taking
% into account only X and Y coordinates.

D = 0;
for i=1:Kd
    x1 = TRAJ1{i}(:,1); y1 = TRAJ1{i}(:,2);
    x2 = TRAJ2_1{i}(:,1); y2 = TRAJ2_1{i}(:,2);
    for k=1:Nt
        vk=[x2(k)-x1(k); y2(k)-y1(k)]; % direction vector
        a=vk(1); b=vk(2); % defining also the orthogonal plane
        d=-(a*Ce(1)+b*Ce(2)); % passing through C
        % parametric line: r_k = P1_k + t*vk | tt defines the projection
        tt = -(a*x1(k)+b*y1(k)+d)/(a*vk(1)+b*vk(2));
        CP = [x1(k);y1(k)]+tt*vk;

        dist = (Ce-CP)'*(Ce-CP);
        D = D + dist;
    end
end
end
```

```
function D = aggrDist2_single(traj1,traj2_1,Ce,Nt)
% aggrDist2_single computes the sum of quadratic distances of Ce from
% the lines defined by the two end-effector positions in demonstrations
% traj1 and traj2 at each of the Nt time instant, taking
% into account only X and Y coordinates.

D = 0;

x1 = traj1(:,1); y1 = traj1(:,2);
x2 = traj2_1(:,1); y2 = traj2_1(:,2);

for k=1:Nt
    vk=[x2(k)-x1(k); y2(k)-y1(k)]; % direction vector
    a=vk(1); b=vk(2); % defining also the orthogonal plane
    d=-(a*Ce(1)+b*Ce(2)); % passing through C
    % parametric line: r_k = P1_k + t*vk | tt defines the projection
    tt = -(a*x1(k)+b*y1(k)+d)/(a*vk(1)+b*vk(2));
    CP = [x1(k);y1(k)]+tt*vk;

    dist = (Ce-CP)'*(Ce-CP);
    D = D + dist;
end
```

```
end
```

B.5 Optimization of surface's parameters

The optimization problems:

- $\underset{a,b,c,d}{\text{argmin}} KL(\mathcal{N}_2||\mathcal{N}_1^s(a,b,c,d))$, for a symmetry plane
- $\underset{x_C,y_C,z_C,R}{\text{argmin}} KL(\mathcal{N}_2||\mathcal{N}_1^s(x_C,y_C,z_C,R))$, for a symmetry sphere
- $\underset{x_C,y_C,R}{\text{argmin}} KL(\mathcal{N}_2||\mathcal{N}_1^s(x_C,y_C,R))$, for a symmetry cylinder

are solved by the three following scripts using MATLAB function `fminunc`. `mu_w1` and `S_w1` are respectively the mean and the covariance for the ProMP$_1$ weigths, `mu_w2`, `S_w2` the mean and the covariance for the ProMP$_2$ weigths, `basis_f` is the matrix that contains the basis functions, `T` the transformation matrix for the change of frame from the first arm base to the second, and the parameters `a0`, `b0`, `c0`, `d0`, `xc0`, `yc0`, `zc0`, `R0` are the ones obtained from initial estimation.

```
% Symmetry plane's parameters optimization
opt = optimoptions(@fminunc,'Display','iter','Algorithm','quasi-newton');
fun = @(X)symm_KLcost(mu_w1,S_w1,mu_w2,S_w2,basis_f,X,T);
plane_new = fminunc(fun,[a0,b0,c0,d0]',opt);
```

```
% Symmetry sphere's parameters optimization
opt = optimoptions(@fminunc,'Display','iter','Algorithm','quasi-newton');
fun = @(X)sphericSymm_KLcost(mu_w1,S_w1,mu_w2,S_w2,basis_f,X,T);
plane_new = fminunc(fun,[xc0,yc0,zc0,R0]',opt);
```

```
% Symmetry cylinder's parameters optimization
opt = optimoptions(@fminunc,'Display','iter','Algorithm','quasi-newton');
fun = @(X)cylindricSymm_KLcost(mu_w1,S_w1,mu_w2,S_w2,basis_f,X,T);
plane_new = fminunc(fun,[xc0,yc0,R0]',opt);
```

`symm_KLcost` is used to get the Kullback-Leibler divergence between the ProMP of the second arm and the symmetric of the first, with respect to a given plane.

```
function c = symm_KLcost(mu_w1,S_w1,mu_w2,S_w2,basis_f,plane,T)
% symm_KLcost KL divergence (mu_w2,S_w2 || mu_w1_symm,S_w1_symm)
%
%   c = symm_KLcost(mu_w1,S_w1,mu_w2,S_w2,basis_f,plane,center,T) computes
%   KL divergence between the 2nd ProMP and the symmetric of the 1st ProMP
%   with respect to the plane in 3D space defined by the parameters a,b,c,d
%   given in 'plane'.
%   [This function uses symm_ProMP and KLdiv_gauss]
```

```
[mu_w1_symm,S_w1_symm] = symm_ProMP(mu_w1,S_w1,basis_f,plane,T);
c = KLdiv_gauss(mu_w2,S_w2,mu_w1_symm,S_w1_symm);
end
```

`sphericSymm_KLcost` is used to get the Kullback-Leibler divergence between the ProMP of the second arm and the symmetric of the first, with respect to a given sphere.

```
function c = sphericSymm_KLcost(mu_w1,S_w1,mu_w2,S_w2,basis_f,param,T)
% sphericSymm_KLcost KL divergence (mu_w2,S_w2 || mu_w1_symm,S_w1_symm)
%
%   c = sphericSymm_KLcost(mu_w1,S_w1,mu_w2,S_w2,basis_f,plane,center,T)
%   computes KL divergence between the 2nd ProMP and the symmetric of the
%   1st ProMP with respect to the sphere defined by the parameters
%   given in 'param'.
%   [This function uses symm_ProMP and KLdiv_gauss]

[mu_w1_symm,S_w1_symm] = sphericSymm_ProMP(mu_w1,S_w1,basis_f,param,T);
c = KLdiv_gauss(mu_w2,S_w2,mu_w1_symm,S_w1_symm);
end
```

`cylindricSymm_KLcost` is used to get the Kullback-Leibler divergence between the ProMP of the second arm and the symmetric of the first, with respect to a given cylinder.

```
function c = cylindricSymm_KLcost(mu_w1,S_w1,mu_w2,S_w2,basis_f,param,T)
% cylindricSymm_KLcost KL divergence (mu_w2,S_w2 || mu_w1_symm,S_w1_symm)
%
%   c = sphericSymm_KLcost(mu_w1,S_w1,mu_w2,S_w2,basis_f,plane,center,T)
%   computes KL divergence between the 2nd ProMP and the symmetric of the
%   1st ProMP with respect to the vertical cylinder defined by the
%   parameters  given in 'param'.
%   [This function uses symm_ProMP and KLdiv_gauss]

[mu_w1_symm,S_w1_symm] = cylindricSymm_ProMPframe2(mu_w1,S_w1,basis_f,param,T);
c = KLdiv_gauss(mu_w2,S_w2,mu_w1_symm,S_w1_symm);
end
```

`KLdiv_gauss` computes the Kullback-Leibler divergence for two Gaussian distributions.

```
function divKL = KLdiv_gauss(mu0,Sigma0,mu1,Sigma1)
% divKL = KLdiv_gauss(mu1,Ssigma1,mu2,Sigma2) Kullback-Leibler (KL)
% divergence for Gaussian distributions.
%
%   KL divergence is a measure of how one probability distribution diverges
%   from a second, expected probability distribution.
%
%   N(mu0,Sigma0): typically represents the "true" distribution of data,
%      observations, or a precisely calculated theoretical distribution
%   N(mu1,Sigma1): typically represents a theory, model, description, or
%      approximation of the first normal distribution.
```

```
k = length(mu0); % Dimension of the normal distribution

divKL = 1/2*(trace(Sigma1^(-1)*Sigma0)+(mu1-mu0)'*Sigma1^(-1)*(mu1-mu0) ...
         -k + log(det(Sigma1)/det(Sigma0)));
end
```

B.6 Reinforcement Learning with REPS

The following script gives a general example of REPS update applied with the **Symmetrization Learning** approach. `mu_w1` and `S_w1` are mean and covariance of ProMP$_1$, `a`, `b`, `c` and `d` are the parameters of the symmetry plane, `lambda` is the parameter used in the covariance regularization term, `Psi` the basis functions block matrix used to compute rollouts from sampled weights. It is necessary to define the function evaluating rewards, that depends from the task to be learned. This framework could be easily adapted to other learning approaches, like **Double Learning** and **Robust Symmetric Learning**.

```
% ProMP update using REPS - Symmetric Learning approach
Kupdates=100;
for epoch=1:Kupdates+1

Kl = 10; % # of roll-outs evaluated
  for i=1:Kl
    % Sample weights vector
    w1_ro(:,i) = mvnrnd(mu_w1,S_w1)';

    % Obtain trajectories
    traj1_ro(:,i) = Psi'*w1_ro(1:3*M,i);
    traj1_xyz = [traj1_ro(1:Nt,i)';traj1_ro(Nt+1:2*Nt,i)';traj1_ro(2*Nt+1:3*Nt,i)'];
    % Change of frame for symm. trajectories
    for t=1:Nt
     [traj1symm_xyz(:,t),~]=symm_3Dgauss(traj1_xyz(:,t),eye(3),a,b,c,d);
    end
    traj1symm_ro_1(:,i)= ...
        [traj1symm_xyz(1,:)';traj1symm_xyz(2,:)';traj1symm_xyz(3,:)'];

    % Compute reward
    R(i) = % REWARD EVALUATING FUNCTION to be defined depending
       % on the task robots must learn

  end

  % Normalized REPS weigths
  dw=REPSupdate(R,0.5,options);
  dw = dw./sum(dw);

  % UPDATE
  mu_learn_update = zeros(size(mu_learn));
  S_learn_update = zeros(size(S_learn));
```

```
  for i=1:Kl
    mu_learn_update =  mu_learn_update + dw(i)*wlearn_ro(:,i);
    S_learn_update =  S_learn_update + ...
        dw(i)*(wlearn_ro(:,i)-mu_learn)*(wlearn_ro(:,i)-mu_learn)'/(Kl-1);
  end

  mu_learn = mu_learn_update;
  S_learn = S_learn_update+eye(size(S_learn))*lambda*0.99^epoch;
end
```

`REPSudpate` returns the relative weights needed to update the policy at each learning iteration.

```
function dw=REPSupdate(REWARDS,epsilon,options)
% epsilon is the KL divergence bound (common value 0.5)
% REWARDS is a row matrix with negative rewards (more negative -> worse)

%% REPS
etamin=0.0005;
etamax=100;

% Get weights with REPS
dualFunctionActual = @(eta_) dualfunction(eta_, REWARDS, epsilon);
eta2 = fmincon(dualFunctionActual,0.01,[], [], [], [], etamin, etamax,[],options);

if or (eta2==etamin,eta2==etamax)
    warning('Eta in its boundary')
end

dw = exp((REWARDS-max(REWARDS)*ones(size(REWARDS)))/eta2)';
% dw: relative weights
end
```

`dualfunction` computes the value of the REPS dual function.

```
function [g] = dualfunction(eta,REWARDS,epsilon)

n_batch = length(REWARDS);
g=epsilon*eta+eta*(log(sum(exp((REWARDS-max(REWARDS))./eta))/n_batch))+max(REWARDS);
if imag(g)>1e-15
   warning('Dual function with imaginary part')
end
end
```